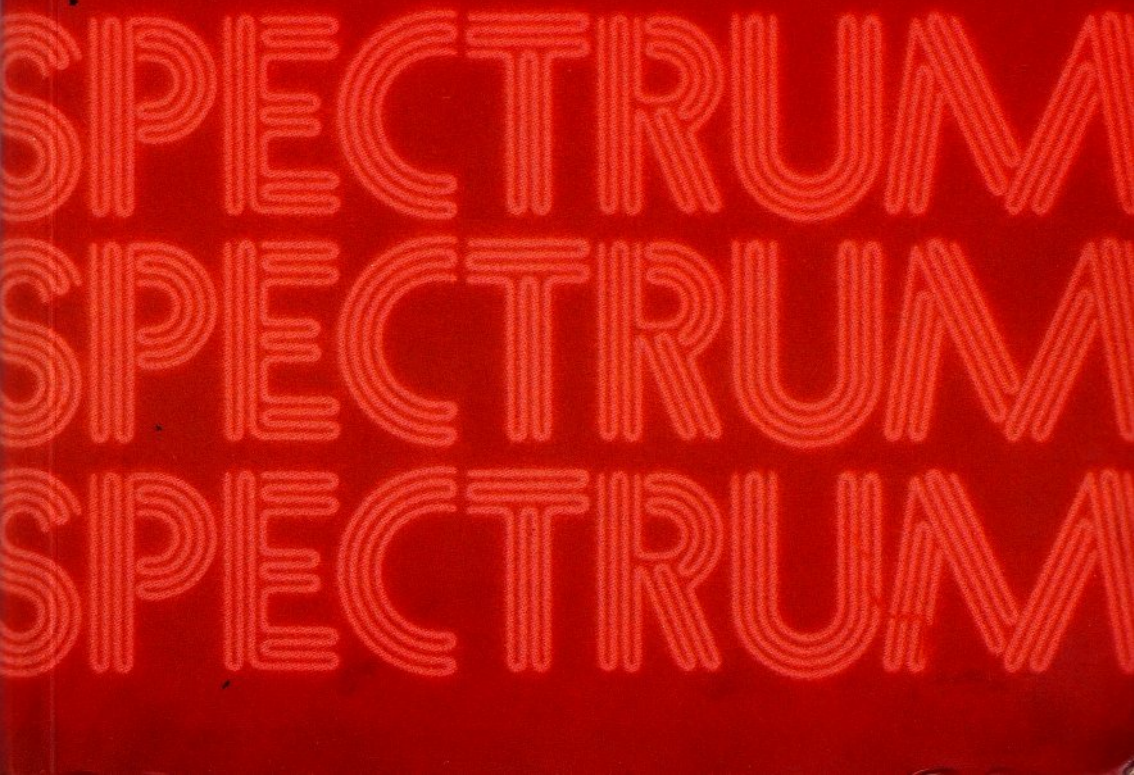
SPECTRUM ADVANCED GRAPHICS WORKBOOK

Marcus D. Bowman



Spectrum Advanced Graphics Workbook

Marcus D. Bowman

McGRAW-HILL Book Company (UK) Limited

London · New York · St Louis · San Francisco · Auckland · Bogotá
Guatemala · Hamburg · Johannesburg · Lisbon · Madrid
Mexico · Montreal · New Delhi · Panama · Paris · San Juan · São
Paulo · Singapore · Sydney · Tokyo · Toronto

Published by
McGRAW-HILL Book Company (UK) Limited
MAIDENHEAD · BERKSHIRE · ENGLAND

British Library Cataloguing in Publication Data

Bowman, Marcus D.
Spectrum advanced graphics workbook.
1. Sinclair ZX Spectrum (Computer)
I. Title
001.64'04 QA76.8.S625

ISBN 0-07-084747-9

Library of Congress Cataloging in Publication Data

Bowman, Marcus D.
Spectrum advanced graphics workbook.

Includes index.
1. Computer graphics. 2. Sinclair ZX Spectrum (Computer)—Programming. I. Title.
T385.B69 1984 001.64'43 84-20153
ISBN 0-07-084747-9

12345 CUP 8765

Typeset by TC Photo-Typesetters, Maidenhead, England

Printed in Great Britain at the University Press, Cambridge

CONTENTS

INTRODUCTION

This book shows you how to create many unusual and rather neat graphics effects on your Spectrum. There are also two complete graphics systems which you can use to create and display objects. One of these is based on character graphics, and the other on a rather powerful shape description system.

No attempt is made to teach BASIC programming, so you should have a working knowledge of Sinclair BASIC. You do not have to be an expert programmer, but you should be able to understand simple program listings and be able to write some simple programs for yourself. You may pick up a few tips as we go along, of course. . .

Similarly, there is no attempt to repeat the elementary graphics programming techniques explained in the Sinclair manual.

There are a large number of graphics routines in this book which can be added to your BASIC programs. Most of the routines are really machine code, but you do not need to understand machine code, because these routines are presented in the form of DATA statements which can be used from within a BASIC program. There are lots of examples to show you exactly how to use each routine. Experiment with the routines; add them to your own programs; and see how you can easily add a professional touch to your graphics. Perhaps you will find out how to achieve that effect you have always wished you could use. . .

Using the machine code routines

All the machine code routines are RELOCATABLE. That means they can be placed at any suitable position in memory and they will still work. However, the routines will all be used along with a BASIC program, so it is important that the program and the routines are stored in separate areas of memory. The Spectrum keeps track of the amount of memory a BASIC program can use by setting RAMTOP to indicate the highest memory location available to the program. The value of RAMTOP is stored in locations 23730 and 23731, and it may be altered by using POKE commands or by using the CLEAR command.

In a Spectrum, the highest memory location available to BASIC is normally:

(48K) 65367 (16K) 32599

Check this on your own Spectrum, using

```
PRINT PEEK 23730 + 256*PEEK 23731
```

Where a routine is given as DATA, the length will also be given. Each routine can be placed in memory above RAMTOP (where it cannot interfere with a BASIC program) by following these steps:

1. Subtract the length of the routine from RAMTOP, to find a new value of RAMTOP.
2. CLEAR to the value from (1)
3. Add 1 to the new value of RAMTOP, to find the address at which the routine starts.
4. POKE the data above RAMTOP, starting from the address found in (3).

For example:
Given a routine of length 39 bytes.

	(48K)	(16K)
1.	New RAMTOP = 65367 − 39 = 65328	New RAMTOP = 32599 − 39 = 32560
2.	CLEAR 65328	CLEAR 32560
3.	Start = RAMTOP + 1 = 65328 + 1 = 65329	Start = RAMTOP + 1 = 32560 + 1 = 32561
4.	POKE the data into memory, starting at 65329	POKE the data into memory, starting at 32561

ENTERING THE ROUTINE

Any of the machine code routines can be used by referring to the starting location, for example:

(48K)

```
RANDOMIZE USR 65329
```

(16K)

```
RANDOMIZE USR 32561
```

This executes the routine which starts at that address. Each routine automatically returns to BASIC when it has finished.

Example

The following routine inverts the screen by swapping the PAPER and INK colours at each PRINT position.

SWAP PAPER AND INK

Changes colours; INK to PAPER, and PAPER to INK

LENGTH 39 bytes

```
DATA 17, 0, 88, 14, 11, 6, 64, 26, 230, 7
DATA 111, 26, 15, 15, 15, 230, 7, 103, 26, 15
DATA 15, 15, 230, 248, 181, 7, 7, 7, 230, 248
DATA 180, 18, 19, 16, 228, 13, 32, 223, 201
```

Here is a short program (I) showing how to use the routine:

(48K)

```
 10 REM SWAP PAPER and INK
 20 REM set colours
 30 PAPER 5: INK 2: BORDER 6
 40 CLS
100 REM set new ramtop
110 CLEAR 65329: REM 16K=32560
120 LET start=65330: REM 16K=32
561
130 LET s=start
140 LET length=39
150 FOR i=1 TO length
160 READ n
170 POKE s,n
180 LET s=s+1
190 NEXT i
200 REM print message
210 FOR i=0 TO 21
220 PRINT AT i,12;"GRAPHICS"
230 NEXT i
300 REM use the routine
310 FOR i=1 TO 20
320 RANDOMIZE USR start
330 PAUSE 50
340 NEXT i
400 REM finished
410 PAPER 7: INK 0: BORDER 7
1000 REM SWAP PAPER AND INK
1010 DATA 17,0,88,14,11,6,64,26,
230,7
```

Program 1 *(continues)*

```
1020 DATA 111,26,15,15,15,230,7,
103,26,15
1030 DATA 15,15,230,248,181,7,7,
7,230,248
1040 DATA 180,18,19,16,228,13,32
,223,201
```

Program I

(16K)

Make the following changes:

```
110 CLEAR 32560
120 LET start = 32561
```

Note the use of the variable 'start'. This simplifies any changes required, especially if the routine is to be used more than once. It also speeds execution of the routine, because BASIC can handle a variable faster than it can a number.

There is one important difference between a BASIC program and a program which may contain machine code routines. When a BASIC program contains an error, it will return control to the operator, eventually. However, if an error occurs while a machine code routine is running, this does not usually happen. Instead, the machine may continue running, but fail to respond to the keyboard, even if the BREAK key is pressed. Sometimes, the Spectrum will execute a NEW command, which can be rather disconcerting!

Because of this, the safest thing to do is to type the program into the Spectrum, and then save it to tape, BEFORE RUNNING THE PROGRAM. Then, if the program crashes fatally, you will be able to load a fresh copy of the program immediately. When errors do occur like this, the most usual cause is a typing error when entering the data containing the machine code routine, so this should be checked carefully before looking elsewhere.

Even if a machine code routine causes the Spectrum to crash, there is no way in which a program of any sort can cause damage to the machine. In the end, YOU remain in control, because you can always switch off!

The object of this book is to provide you with a series of routines which will allow you to increase the quality of your graphics. These routines are meant to add to the fun you can have with your Spectrum, so enjoy yourself!

1 PRINTING

The screen

The Spectrum uses a MEMORY-MAPPED display. This means that the contents of the screen represent the contents of the section of memory from 16384 to 22527. The area from 22528 to 23295 contains the attributes of the screen, e.g., PAPER, INK, FLASH, and BRIGHT. Memory-mapping is a useful technique, because it ensures that the display always occupies a known area of memory. This allows direct manipulation of the screen contents by manipulating the corresponding area of memory.

The section of memory corresponding to the screen contents is called the DISPLAY FILE. The display file holds a number of BYTES, each consisting of 8 BITS, or binary digits. The bits correspond exactly to the individual points or PIXELS which can be displayed on the screen.

The screen is 32 bytes wide. Since each byte contains 8 bits, there are $32 \times 8 = 256$ pixel positions across the screen. Vertically, there are 192 rows, providing 192 pixels from top to bottom. Figure 1.1 shows the layout of the bytes which make up the screen.

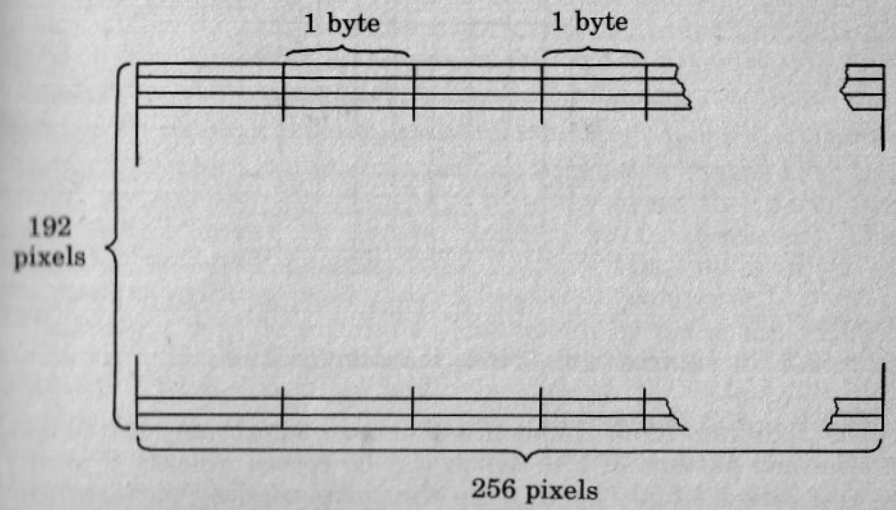


Figure 1.1 The screen layout

A pixel, then, is the smallest individual point on the screen which can be addressed, and corresponds in size to a full stop. Each pixel may be

LIT (ON) when the corresponding bit is set to 1
UNLIT (OFF) when the corresponding bit is set to 0

The top left 8 pixels reflect the state of the 8 bits in the byte stored in location 16384. So,

```
PAPER 7: INK 0: CLS
POKE 16384,2
```

places the binary number 00000010 in 16384. This will display a lit pixel at plot co-ordinates 6,175

```
POKE 16384,0
```

will ensure that all 8 pixels in the top left of the screen are unlit.

Printing

So that the Spectrum can handle the task of printing characters on the screen quickly and efficiently, the PRINT command deals with pre-defined sections of the screen.

In the *x* direction, each print position corresponds to a byte. In the *y* direction, each print position is 8 bytes high. So, each print position contains 8 × 8 = 64 pixels, as shown in Fig. 1.2.

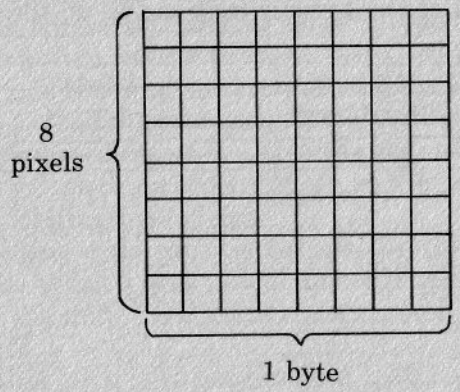


Figure 1.2 The organization of pixels in a print position

The Spectrum ROM contains a character set already stored as a pre-defined pattern of bits which can be copied quickly into any print position simply by copying the corresponding pattern from ROM into the display file.

This character set occupies locations 15616 – 16383 ($3D00 – $3DFF) and holds the bit patterns for 96 characters. Each character occupies 8 bytes.

The second character occupies locations 15624 – 15631, and can be examined by peeking the appropriate location, then changing the resulting value into binary.

	decimal	binary
PRINT PEEK 15624 results in	0	= 00000000
PRINT PEEK 15625 results in	16	= 00010000
PRINT PEEK 15626 results in	16	= 00010000
PRINT PEEK 15627 results in	16	= 00010000
PRINT PEEK 15628 results in	16	= 00010000
PRINT PEEK 15629 results in	0	= 00000000
PRINT PEEK 15630 results in	16	= 00010000
PRINT PEEK 15631 results in	0	= 00000000

Studying the pattern of 1s in the binary numbers above shows which character is stored in locations 15624 – 15631!!!!

The graphics characters which may be typed on the keyboard are not stored in the same way. Instead, the necessary bit patterns are constructed as required and stored in the display file when they are to be displayed.

There is also a section of memory which can hold up to 21 user-definable character bit patterns. The address of the start of this area may vary, and is found by using

```
PRINT PEEK 23675 + 256*PEEK 23676
```

The normal results are

(48K)	(16K)
65368	32600

Holding bit patterns in memory like this is a convenient way of dealing with the printable characters, because it means that the BASIC programmer does not have to worry about the individual pixels which must be lit to produce each character. The disadvantage is that printing normally only takes place in one of 704 possible printing positions (22 lines of 32 columns).

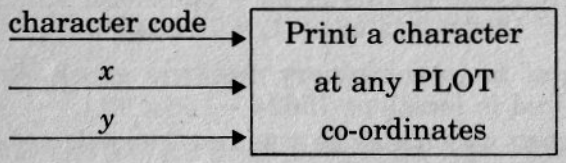
Characters may be printed on the screen by using the PRINT command with the TAB, comma, semicolon, and AT modifiers if required. Characters may also be printed by specifying the appropriate character code. These character codes are listed in the Spectrum manual. The usable codes are 32 (space) to 127 (©) – the 'normal' character set, 128 to 143 – the pre-programmed graphics characters, and 144 to 164 (the user-defined graphics characters).

PRINT CHR$ 65 prints an A
because the code for that character is 65.

Sometimes, it would be useful to be able to print anywhere on the screen, starting at any pixel position, and crossing the boundaries of the normal print positions, if necessary.

The CHARACTER PLOTTER routine may be used instead of the normal PRINT command, and it allows characters to be printed beginning at any PLOT co-ordinates on the screen simply by specifying the x and y co-ordinates of the bottom left of the 8 × 8 character block, together with the appropriate character code.

CHARACTER PLOTTER



LENGTH 203 bytes

Using the routine

POKE the character code into location 23296.
POKE the x co-ordinate of the bottom left corner of the character into location 23297.
POKE the y co-ordinate of the bottom left corner of the character into location 23298.

Note: The routine will not plot a character which would lie wholly or partly off the screen.

Any character may be plotted, except the graphics characters (codes 128–143). User-defined characters may be plotted.

Character plotter data

```
58, 2, 91, 254, 32, 56, 76, 254, 128, 56,

10, 254, 144, 56, 68, 254, 165, 56, 9, 24,

62, 33, 7, 61, 214, 32, 24, 6, 42, 123,

92, 43, 214, 143, 17, 8, 0, 254, 0, 40,

4, 25, 61, 24, 248, 6, 0, 126, 43, 229,
```



```
14, 0, 24, 1, 241, 7, 245, 197, 245, 33,

0, 91, 126, 254, 249, 48, 18, 129, 79, 35,

126, 254, 169, 48, 10, 128, 71, 24, 8, 24,

229, 24, 220, 24, 117, 24, 111, 62, 16, 128,

71, 33, 0, 64, 254, 128, 48, 9, 17, 0,

8, 25, 254, 64, 48, 1, 25, 203, 184, 203,

176, 62, 63, 144, 17, 32, 0, 254, 8, 56,

5, 214, 8, 25, 24, 247, 254, 0, 40, 4,

61, 36, 24, 248, 121, 254, 8, 56, 5, 214,

8, 35, 24, 247, 79, 62, 7, 145, 55, 63,

87, 71, 94, 254, 0, 40, 4, 203, 11, 16,

252, 241, 56, 4, 203, 131, 24, 2, 203, 195,

122, 66, 254, 0, 40, 4, 203, 3, 16, 252,

115, 193, 12, 121, 254, 8, 32, 147, 4, 241,

225, 120, 254, 8, 32, 141, 40, 4, 241, 193,

241, 225, 201
```

Example 1.1

```
 10 REM PLOT CHARACTERS
 20 REM set colours
 30 PAPER 4: INK 7: BORDER 4
 40 CLS
100 REM set new ramtop
110 CLEAR 65164: REM 16K=32396
120 LET start=65165: REM 16K=32
397
200 REM place routine
210 LET length=203
220 LET s=start
230 FOR i=1 TO length
```

Program 1.1 *(continues)*

```
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 300 REM print a name
 310 INPUT "Type your first name
";n$
 320 REM set start x,y
 330 LET x=0
 340 LET y=0
 350 REM print characters
 360 FOR i=1 TO LEN n$
 370 REM poke start x and y
 380 POKE 23296,x
 390 POKE 23297,y
 400 REM poke character code
 410 POKE 23298,CODE n$(i)
 420 REM execute routine
 430 RANDOMIZE USR start
 440 REM change x and y
 450 LET x=x+8
 460 LET y=y+4
 470 NEXT i
 480 REM restore colours
 490 PAPER 7: INK 0: BORDER 7
 500 REM CHARACTER PLOTTER
 510 DATA 58,2,91,254,32,56,76,2
54,128,56
 520 DATA 10,254,144,56,68,254,1
65,56,9,24
 530 DATA 62,33,7,61,214,32,24,6
,42,123
 540 DATA 92,43,214,143,17,8,0,2
54,0,40
 550 DATA 4,25,61,24,248,6,0,126
,43,229
 560 DATA 14,0,24,1,241,7,245,19
7,245,33
 570 DATA 0,91,126,254,249,48,18
,129,79,35
 580 DATA 126,254,169,48,10,128,
71,24,8,24
 590 DATA 229,24,220,24,117,24,1
11,62,16,128
 600 DATA 71,33,0,64,254,128,48,
9,17,0
 610 DATA 8,25,254,64,48,1,25,20
3,184,203
 620 DATA 176,62,63,144,17,32,0,
254,8,56
 630 DATA 5,214,8,25,24,247,254,
0,40,4
 640 DATA 61,36,24,248,121,254,8
,56,5,214
 650 DATA 8,35,24,247,79,62,7,14
```



```
5,55,63
 660 DATA 87,71,94,254,0,40,4,20
3,11,16
 670 DATA 252,241,56,4,203,131,2
4,2,203,195
 680 DATA 122,66,254,0,40,4,203,
3,16,252
 690 DATA 115,193,12,121,254,8,3
2,147,4,241
 700 DATA 225,120,254,8,32,141,4
0,4,241,193
 710 DATA 241,225,201
```

Program 1.1

Experiment with the starting co-ordinates (lines 380, 390) and the amount by which x and y are varied between characters (lines 450, 460).

To erase a character using CHARACTER PLOTTER, print a 'space' character (code 32) over it, using the same routine.

CHARACTER PLOTTER is a useful routine, but it is restricted to plotting characters of the same size, shape, and orientation as those in the inbuilt character set, and the user-defined characters.

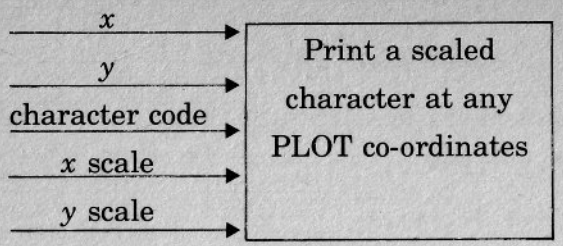
SCALED CHARACTER PLOTTER overcomes the limitations on the character size. Using this routine, scale factors may be set independently for the x and y directions, from 1 upwards. A scale factor of 1 corresponds to the normal size of character; 2 produces a double sized character, and so on. Since the x and y scales may be different, the style of the characters may be changed.

x scale = 1 and y scale = 20
produces a tall thin character
x scale = 10 and y scale = 5
produces a squat character.

The practical limits on the scales are:
x scale = 32 (since the maximum number of pixels is $8 \times 32 = 256$, in the x direction).
y scale = 24 (since the maximum number of pixels is $8 \times 24 = 192$, in the y direction).

SCALED CHARACTER PLOTTER will plot any character except the graphics characters (but it will plot the user-defined characters).

SCALED CHARACTER PLOTTER



LENGTH 301 bytes

Using the routine

POKE the *x* co-ordinate of the bottom left corner of the character into location 23296.
POKE the *y* co-ordinate of the bottom left corner of the character into location 23297.
POKE the character code into location 23298.
POKE the *x* scale into location 23299.
POKE the *y* scale into location 23300.

Note: Unlike the CHARACTER PLOTTER routine, SCALED CHARACTER PLOTTER does not check to see whether the character lies off the screen.

Scaled character plotter data

```
58, 2, 91, 254, 32, 56, 103, 254, 128, 56,

10, 254,144, 56, 95, 254, 165, 56, 9, 24,

89, 33, 7, 61, 214, 32, 24, 6, 42, 123,

92, 43, 214, 143, 17, 8, 0, 254, 0, 40,

4, 25, 61, 24, 248, 229, 33, 0, 91, 126,

79, 35, 126, 71, 225, 229, 33, 4, 91, 126,

35, 35, 119, 33, 8, 91, 62, 0, 119, 225,

24, 22, 241, 193, 225, 43, 229, 33, 4, 91,

126, 35, 35, 119, 225,24, 3, 241, 193, 225,

58, 0, 91, 79, 126, 229, 197, 245, 33, 7,
```



```
91, 62, 0, 119, 241, 7, 245, 33, 3, 91,

126, 35, 35, 119, 24, 8,24, 103, 24, 208,

24, 221, 24, 236, 62, 16, 128, 71, 33,0,

64, 254, 128, 48, 9, 17, 0, 8, 25, 254,

64, 48, 1, 25, 203, 184, 203, 176, 62, 63,

144, 17, 32, 0, 254, 8, 56, 5, 214, 8,

25, 24, 247, 254, 0, 40, 4, 61, 36, 24,

248, 121, 254, 8, 56, 5, 214, 8, 35, 24,

247, 79, 62, 7, 145, 55, 63, 87, 71, 94,

254, 0, 40, 4, 203, 11, 16, 252, 241, 245,

56, 4, 203, 131, 24, 2, 203, 195, 122, 66,

254, 0, 40, 4, 203, 3, 16, 252, 115, 24,

8, 24, 77, 24, 149, 24, 149, 24, 149, 33,

5, 91, 126, 61, 119, 254, 0, 40, 7, 241,

193, 12, 197, 245, 24, 134, 33, 7, 91, 126,

60, 119, 254, 8, 40, 7, 241, 193, 12, 197,

245, 24, 220, 33, 6, 91, 126, 61, 119, 254,

0, 40, 7, 241, 193, 4, 197, 245, 24, 201,

33, 8, 91, 126, 60, 119, 254, 8, 40, 7,

241, 193, 4, 197, 245, 24, 182, 241, 193, 225,

201
```

Example 1.2

```
  10 REM SCALED CHARACTERS
  20 REM set colours
  30 PAPER 6: INK 1: BORDER 6
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65066: REM 16K=32298
 120 LET start=65067: REM 16K=32
299
 200 REM place routine
 210 LET length=301
 220 LET s=start
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 300 REM fetch a name
 310 INPUT "Type your first name
";n$
 320 REM set start x,y
 330 LET x=88
 340 LET y=65
 350 POKE 23296,x: POKE 23297,y
 360 REM set scales
 370 LET xscale=10
 380 LET yscale=7
 390 POKE 23299,xscale: POKE 233
00,yscale
 400 REM display characters
 410 FOR i=1 TO LEN n$
 420 REM poke code
 430 POKE 23298,CODE n$(i)
 440 REM execute routine
 450 RANDOMIZE USR start
 460 PAUSE 50
 470 REM print space (code=32)
 480 POKE 23298,32
 490 RANDOMIZE USR start
 500 NEXT i
 510 REM restore colours
 520 PAPER 7: INK 0: BORDER 7
 530 CLS
 600 REM SCALED CHARACTER PLOTTE
R
 610 DATA 58,2,91,254,32,56,103,
254,128,56
 620 DATA 10,254,144,56,95,254,1
65,56,9,24
 630 DATA 89,33,7,61,214,32,24,6
,42,123
 640 DATA 92,43,214,143,17,8,0,2
54,0,40
 650 DATA 4,25,61,24,248,229,33,
```



```
0,91,126
 660 DATA 79,35,126,71,225,229,3
3,4,91,126
 670 DATA 35,35,119,33,8,91,62,0
,119,225
 680 DATA 24,22,241,193,225,43,2
29,33,4,91
 690 DATA 126,35,35,119,225,24,3
,241,193,225
 700 DATA 58,0,91,79,126,229,197
,245,33,7
 710 DATA 91,62,0,119,241,7,245,
33,3,91
 720 DATA 126,35,35,119,24,9,24,
103,24,208
 730 DATA 24,221,24,236,62,16,12
0,71,33,0
 740 DATA 64,254,128,48,9,17,0,8
,25,254
 750 DATA 64,48,1,25,203,184,203
,176,62,63
 760 DATA 144,17,32,0,254,8,56,5
,214,8
 770 DATA 25,24,247,254,0,40,4,6
1,36,24
 780 DATA 248,121,254,8,56,5,214
,0,35,24
 790 DATA 247,79,62,7,145,55,63,
87,71,94
 800 DATA 254,0,40,4,203,11,16,2
52,241,245
 810 DATA 56,4,203,131,24,2,203,
195,122,66
 820 DATA 254,0,40,4,203,3,16,25
2,115,24
 830 DATA 8,24,77,24,149,24,149,
24,149,33
 840 DATA 5,91,126,61,119,254,0,
40,7,241
 850 DATA 193,12,197,245,24,134,
33,7,91,126
 860 DATA 60,119,254,8,40,7,241,
193,12,197
 870 DATA 245,24,220,33,6,91,126
,61,119,254
 880 DATA 0,40,7,241,193,4,197,2
45,24,201
 890 DATA 33,8,91,126,60,119,254
,0,40,7
 900 DATA 241,193,4,197,245,24,1
82,241,193,225
 910 DATA 201
```

Program 1.2

Varying the starting co-ordinates (lines 330, 340) and the x and y scales (lines 370, 380) allows a variety of effects to be achieved. Removing lines 470, 480, 490 shows that one character will overprint another, provided the scales are the same, although the effect is rather different than when overprinting with a space character.

The two previous routines provide a great deal of extra control over the positioning and size of characters placed on the screen. However, another useful addition is the ability to rotate a character before it is placed on the screen. ROTATED SCALED CHARACTER PLOTTER allows a rotation to be specified, in steps of 90° (clockwise), as shown in Fig. 1.3.

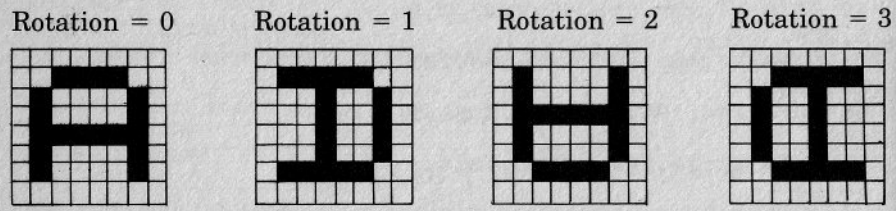
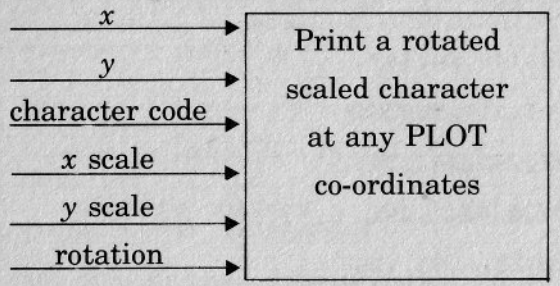


Figure 1.3 The effect of different values of rotation

This is useful for producing vertical scales (on graphs, for example), or for rotating user-defined characters.

ROTATED SCALED CHARACTER PLOTTER



LENGTH 414 bytes

Using the routine

POKE the x co-ordinate of the bottom left corner of the character into location 23296.
POKE the y co-ordinate of the bottom left corner of the character into location 23297.
POKE the character code into location 23298.

POKE the x scale into location 23299.
POKE the y scale into location 23300.
POKE the rotation into location 23301.

Note: The rotation must be in the range 0–3.

The routine uses locations 23302–23313 inclusive, as workspace.

Rotated scaled character plotter data

```
58, 5, 91, 254, 4, 48, 43, 58, 2, 91,

254, 32, 56, 36, 254, 128, 56, 8, 254, 144,

56, 28, 254, 165, 56, 7, 33, 7, 61, 214,

32, 24, 6, 42, 123, 92, 43, 214, 143, 17,

8, 0, 254, 0, 40, 6, 25, 61, 24, 248,

24, 104, 58, 5, 91, 254, 0, 32, 10, 17,

17, 91, 1, 8, 0, 237, 184, 24, 82, 254,

2, 32, 25, 6, 8, 17, 10, 91, 197, 26,

79, 126, 6, 8, 23, 203, 25, 16, 251, 121,

18, 193, 19, 43, 16, 238, 24, 53, 254, 3,

32, 25, 6, 8, 17, 17, 91, 126, 197, 6,

8, 23, 235, 78, 203, 25, 113, 235, 27, 16,

246, 193, 43, 16, 235, 24, 24, 6, 8, 17,

17, 91, 126, 197, 6, 8, 203, 31, 235, 78,

203, 17, 113, 235, 27, 16, 245, 193, 43, 16,

234, 33, 17, 91, 229, 24, 2, 24, 70, 33,

0, 91, 126, 79, 35, 126, 71, 33, 4, 91,

126, 35, 35, 35, 119, 35, 35, 62, 0, 119,
```

```
225, 24, 23, 241, 193, 225, 43, 229, 33, 4,

91, 126, 35, 35, 35, 119, 225, 24, 3, 241,

193, 225, 58, 0, 91, 79, 126, 229, 197, 245,

33, 8, 91, 62, 0, 119, 241, 7, 245, 33,

3, 91, 126, 35, 35, 35, 119, 24, 8, 24,

103, 24, 206, 24, 220, 24, 235, 62, 16, 128,

71, 33, 0, 64, 254, 128, 48, 9, 17, 0,

8, 25, 254, 64, 48, 1, 25, 203, 184, 203,

176, 62, 63, 144, 17, 32, 0, 254, 8, 56,

5, 214, 8, 25, 24, 247, 254, 0, 40, 4,

61, 36, 24, 248, 121, 254, 8, 56, 5, 214,

8, 35, 24, 247, 79, 62, 7, 145, 55, 63,

87, 71, 94, 254, 0, 40, 4, 203, 11, 16,

252, 241, 245, 56,4, 203, 131, 24, 2, 203,

195, 122, 66, 254, 0, 40, 4, 203, 3, 16,

252, 115, 24, 8, 24, 77, 24, 149, 24, 149,

24, 149, 33, 6, 91, 126, 61, 119, 254, 0,

40, 7, 241, 193, 12, 197, 245, 24, 134, 33,

8, 91, 126, 60, 119, 254, 8, 40, 7, 241,

193, 12, 197, 245, 24, 220, 33, 7, 91, 126,

61, 119, 254, 0, 40, 7, 241, 193, 4, 197,

245, 24, 201, 33, 9, 91, 126, 60, 119, 254,
```



```
8, 40, 7, 241, 193, 4, 197, 245, 24, 182,

241, 193, 225, 201
```

Example 1.3

```
  10 REM ROTATED SCALED CHARACTE
RS
  20 REM set colours
  30 PAPER 6: INK 1: BORDER 6
  40 CLS
 100 REM set new ramtop
 110 CLEAR 64953: REM 16K=32185
 120 LET start=64954: REM 16K=32
186
 200 REM place routine
 210 LET length=414
 220 LET s=start
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 300 REM fetch a name
 310 INPUT "Type your first name
";n$
 320 REM set start x,y
 330 LET x=0
 340 LET y=20
 350 REM set scale
 360 LET xscale=INT (256/8)/LEN
n$
 370 LET yscale=xscale
 380 POKE 23299,xscale: POKE 233
00,yscale
 390 REM rotation
 400 LET r=0
 410 REM print characters
 420 FOR i=1 TO LEN n$
 430 REM set bottom left
 440 POKE 23296,x: POKE 23297,y
 450 REM set rotation
 460 POKE 23301,r
 470 REM set character
 480 POKE 23298,CODE n$(i)
 490 REM execute routine
 500 RANDOMIZE USR start
 510 REM increment x,r
 520 LET x=x+8*xscale
 530 LET r=r+1
 540 IF r>3 THEN  LET r=0
 550 NEXT i
 560 PAUSE 100
```

Program 1.3 *(continues)*

```
 570 REM restore colours
 580 PAPER 7: INK 0: BORDER 7
 590 CLS
 600 REM ROTATED SCALED CHARACTE
R PLOTTER
 610 DATA 58,5,91,254,4,48,43,58
,2,91
 620 DATA 254,32,56,36,254,128,5
6,8,254,144
 630 DATA 56,28,254,165,56,7,33,
7,61,214
 640 DATA 32,24,6,42,123,92,43,2
14,143,17
 650 DATA 8,0,254,0,40,6,25,61,2
4,248
 660 DATA 24,104,58,5,91,254,0,3
2,10,17
 670 DATA 17,91,1,8,0,237,184,24
,82,254
 680 DATA 2,32,25,6,8,17,10,91,1
97,26
 690 DATA 79,126,6,8,23,203,25,1
6,251,121
 700 DATA 18,193,19,43,16,238,24
,53,254,3
 710 DATA 32,25,6,8,17,17,91,126
,197,6
 720 DATA 8,23,235,78,203,25,113
,235,27,16
 730 DATA 246,193,43,16,235,24,2
4,6,8,17
 740 DATA 17,91,126,197,6,8,203,
31,235,78
 750 DATA 203,17,113,235,27,16,2
45,193,43,16
 760 DATA 234,33,17,91,229,24,2,
24,70,33
 770 DATA 0,91,126,79,35,126,71,
33,4,91
 780 DATA 126,35,35,35,119,35,35
,62,0,119
 790 DATA 225,24,23,241,193,225,
43,229,33,4
 800 DATA 91,126,35,35,35,119,22
5,24,3,241
 810 DATA 193,225,58,0,91,79,126
,229,197,245
 820 DATA 33,8,91,62,0,119,241,7
,245,33
 830 DATA 3,91,126,35,35,35,119,
24,8,24
 840 DATA 103,24,206,24,220,24,2
35,62,16,128
 850 DATA 71,33,0,64,254,128,48,
9,17,0
```



```
 860 DATA 8,25,254,64,48,1,25,20
3,184,203
 870 DATA 176,62,63,144,17,32,0,
254,8,56
 880 DATA 5,214,8,25,24,247,254,
0,40,4
 890 DATA 61,36,24,248,121,254,8
,56,5,214
 900 DATA 8,35,24,247,79,62,7,14
5,55,63
 910 DATA 87,71,94,254,0,40,4,20
3,11,16
 920 DATA 252,241,245,56,4,203,1
31,24,2,203
 930 DATA 195,122,66,254,0,40,4,
203,3,16
 940 DATA 252,115,24,8,24,77,24,
149,24,149
 950 DATA 24,149,33,6,91,126,61,
119,254,0
 960 DATA 40,7,241,193,12,197,24
5,24,134,33
 970 DATA 8,91,126,60,119,254,8,
40,7,241
 980 DATA 193,12,197,245,24,220,
33,7,91,126
 990 DATA 61,119,254,0,40,7,241,
193,4,197
1000 DATA 245,24,201,33,9,91,126
,60,119,254
1010 DATA 8,40,7,241,193,4,197,2
45,24,182
1020 DATA 241,193,225,201
```

Program 1.3

The x scale is set in line 360, and is calculated in such a way that the more characters there are to be printed, the smaller the scale. Since the basic character size is 8 pixels wide (when the scale is 1), 256/8 represents maximum x scale when one character is to be plotted. Dividing this by the number of characters to be plotted sets the x scale. The y scale is set to the same as the x scale in this example, simply because this results in a 'square' character. Too few, or too many characters in the word to be printed may cause problems. Try calculating more suitable scale values, for a given word length.

The attributes

An unusual, but rather useful feature of the Spectrum is the way it handles the ATTRIBUTES associated with each of the pre-defined print positions. These attributes are:

FLASH	0 or 1	0 = OFF 1 = ON
BRIGHT	0 or 1	0 = OFF 1 = ON
PAPER	0 to 6	
INK	0 to 6	

Each print position has an attribute byte associated with it, and that byte holds the attributes as shown in Fig. 1.4.

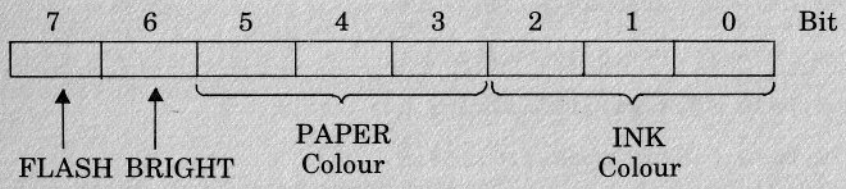


Figure 1.4 Organization of attributes within a byte

The ATTRIBUTE file occupies memory locations 22528 to 23295 inclusive. Within this section of memory, the attribute bytes are set out in order, starting from the top left corner of the screen and following the lines and columns of the print positions on the screen.

Note that only 1 attribute byte is required to hold the attributes for a whole print position of 8 bytes.

The address of any attribute byte can be calculated if the line and column values of the corresponding print position are known.

ADDRESS = 22528 + (line * 32) + column

The attributes for a print position may be set from within a BASIC program using the keywords FLASH, BRIGHT, PAPER, and INK. These set the attribute byte when a character is printed in the corresponding print position. However, the attributes may be set using POKE commands which allow attributes to be set without demanding that a character be printed on the screen. This can be used to change attributes without changing the contents of the display file.

To give the print position at 6,3 attributes

FLASH 1, BRIGHT 0, PAPER 0, INK 6

first write each attribute code as a binary number.

FLASH 1, BRIGHT 0, PAPER 000, INK 110

Next, calculate the total value of the attribute byte, using the bit values shown in Fig. 1.5.

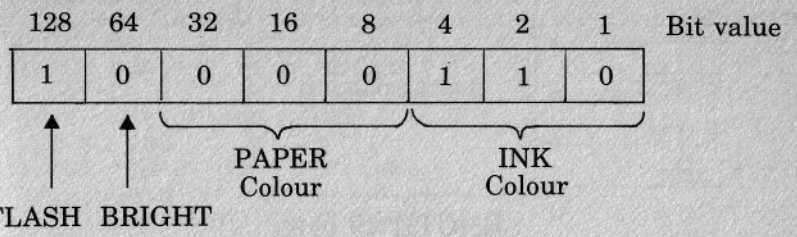


Figure 1.5 Bit values within an attribute byte

Total value = 128 + 4 + 2 = 134

Now calculate the address of the corresponding attribute byte.

ADDRESS = 22528 + (6*32) + 3
= 22723

Set the attribute byte using POKE 22723,134

Printing a character at 6,3 may change the attributes at that position, because the normal printing routine sets the attributes using the permanent attributes (i.e., those set by the last PAPER, INK, FLASH or BRIGHT commands).

However, printing a character then setting the attributes by POKEing a value into the display file will allow the new attribute value to remain at that printing position until another character is printed there. The attributes can be set using POKE even when a character is not to be printed at that position, which is an advantage. Similarly, attributes can be changed without changing the character at a particular position on the screen.

Try this:

```
PRINT AT 6,3;"A"
POKE 22723,134
```

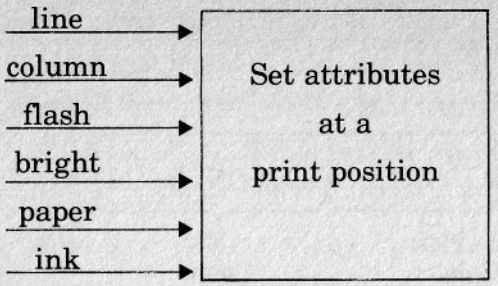
Whatever attributes were used when printing the character will be changed to the attributes with code 134.

Reset the attributes at this position using

POKE 22723,0

The ATTRIBUTE SET routine may be used to calculate the attribute value for a print position and to place the appropriate value in the attribute file.

ATTRIBUTE SET



LENGTH 49 bytes

Using the routine

POKE the line number into location 23296.
POKE the column number into location 23297.
POKE the value of FLASH into location 23298.
POKE the value of BRIGHT into location 23299.
POKE the PAPER number into location 23300.
POKE the INK number into location 23301.

Attribute set data

33, 0, 91, 126, 35, 94, 254, 22, 48, 34,
33, 0, 88, 22, 0, 25, 17, 32, 0, 254,
0, 40, 4, 25, 61, 24, 248, 235, 33, 2,
91, 62, 0, 174, 7, 35, 174, 7, 7, 7,
35, 174, 7, 7, 7, 35, 174, 18, 201

Example 1.4

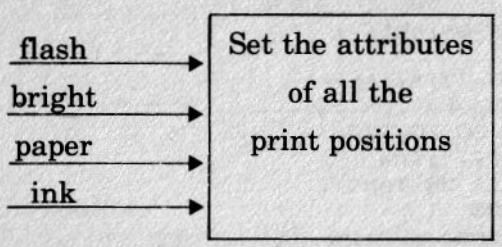
```
 10 REM SET ATTRIBUTES AT PRINT
POSITION
 20 REM set colours
 30 PAPER 7: INK 0: BORDER 7
 40 CLS
100 REM set new ramtop
110 CLEAR 65318: REM 16K=32550
120 LET start=65319: REM 16K=32
551
200 REM place routine
210 LET length=49
220 LET s=start
230 FOR i=1 TO length
240 READ n
250 POKE s,n
260 LET s=s+1
270 NEXT i
300 REM print message
310 PRINT AT 10,7;"Set my attri
butes"
400 REM fetch attributes
410 INPUT "FLASH (1 or 0)";flas
h
420 IF flash<>0 AND flash<>1 TH
EN  GO TO 900
430 INPUT "BRIGHT (1 or 0)";bri
ght
440 IF bright<>0 AND bright<>1
THEN  GO TO 900
450 INPUT "PAPER (1 - 7)";paper
460 IF paper<0 OR paper>7 OR pa
per<>INT paper THEN  GO TO 900
470 INPUT "INK (0 - 7)";ink
480 IF ink<0 OR ink>7 OR ink<>I
NT ink THEN  GO TO 900
500 REM set attributes
510 POKE 23298,flash
520 POKE 23299,bright
530 POKE 23300,paper
540 POKE 23301,ink
550 REM set line
560 POKE 23296,10
570 REM move across columns
580 FOR c=7 TO 23
590 POKE 23297,c
600 REM execute change
610 RANDOMIZE USR start
620 NEXT c
630 REM try again
640 GO TO 400
900 REM finished
1000 REM ATTRIBUTE SET
1010 DATA 33,0,91,126,35,94,254,
22,48,34
1020 DATA 33,0,88,22,0,25,17,32,
0,254
1030 DATA 0,40,4,25,61,24,248,23
5,33,2
1040 DATA 91,62,0,174,7,35,174,7
,7,7
1050 DATA 35,174,7,7,7,35,174,18
,201
```

Program 1.4

Instead of simply setting the attributes of one print position, this example uses a loop at lines 570–620 to move across several print positions, setting the attributes of each. This shows how the basic routine can be used to change an area of the attribute file, and allows a large variety of effects, such as areas of changing colour for highlighting.

The ability to set the attributes of a single print position can be used to set the attributes of all the print positions on the screen. This is of most use when the attributes are all to be changed to similar values.

SET ALL ATTRIBUTES



LENGTH 43 bytes

Using the routine

POKE the value of FLASH into location 23296.
POKE the value of BRIGHT into location 23297.
POKE the PAPER number into location 23298.
POKE the INK number into location 23299.

Set all attributes data

1, 32, 22, 17, 0, 88, 33, 2, 91, 62,
0, 174, 7, 35, 174, 7, 7, 7, 35, 174,
7, 7, 7, 35, 174, 245, 241, 18, 245, 19,
16, 250, 62, 32, 71, 13, 121, 254, 0, 32,
241, 241, 201

Example 1.5

```
 10 REM SET ALL ATTRIBUTES
 20 REM set colours
 30 PAPER 7: INK 0: BORDER 7
 40 CLS
100 REM set new ramtop
110 CLEAR 65324: REM 16K=32556
120 LET start=65325: REM 16K=32
557
200 REM place routine
210 LET length=43
220 LET s=start
230 FOR i=1 TO length
240 READ n
250 POKE s,n
260 LET s=s+1
```



```
270 NEXT i
300 REM print message
310 PRINT AT 10,5;"Set screen a
ttributes"
400 REM fetch attributes
410 INPUT "FLASH (1 or 0)";flas
h
420 IF flash<>0 AND flash<>1 TH
EN  GO TO 900
430 INPUT "BRIGHT (1 or 0)";bri
ght
440 IF bright<>0 AND bright<>1
THEN  GO TO 900
450 INPUT "PAPER (1 - 7)";paper
460 IF paper<0 OR paper>7 OR pa
per<>INT paper THEN  GO TO 900
470 INPUT "INK (0 - 7)";ink
480 IF ink<0 OR ink>7 OR ink<>I
NT ink THEN  GO TO 900
500 REM set attributes
510 POKE 23298,flash
520 POKE 23299,bright
530 POKE 23300,paper
540 POKE 23301,ink
600 REM execute change
610 RANDOMIZE USR start
630 REM try again
640 GO TO 400
900 REM finished
1000 REM SET ALL ATTRIBUTES
1010 DATA 1,32,22,17,0,88,33,2,9
1,62
1020 DATA 0,174,7,35,174,7,7,7,3
5,174
1030 DATA 7,7,7,35,174,245,241,1
8,245,19
1040 DATA 16,250,62,32,71,13,121
,254,0,32
1050 DATA 241,241,201
```

Program 1.5

Attributes may be examined by using the ATTR line, column command, or by employing PEEK to fetch a value from the attribute file. In either case, the result is a decimal number which must be converted to binary before the individual attributes can be recovered.

To decode the attributes at print position 10,12 calculate the address of the attribute byte in the attribute file.

ADDRESS = 22528 + (10*32) + 12
= 22860

LET A = PEEK 22860 assigns the attribute value to *A*

Suppose that *A* has the value 112: changing to a binary number yields 01110000, as shown in Fig. 1.6.

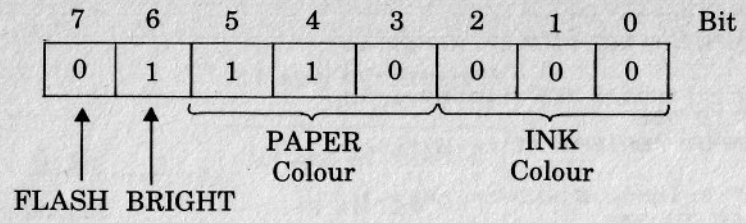


Figure 1.6 The attribute value decoded

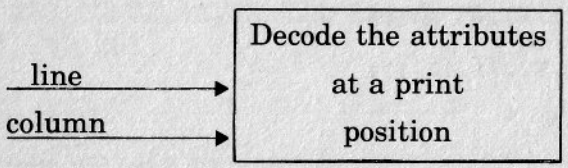
So, the attributes are:

FLASH 0, BRIGHT 1, PAPER 6, INK 0

A much faster method of decoding the attributes at any given print position is provided by the DECODE ATTRIBUTES routine. This routine also has the advantage that it is easy to recover the individual attributes from within a BASIC program and to act on their values, for example,

If a print position has GREEN paper, turn it RED (using the ATTRIBUTE SET routine if desired), if not, then turn the paper BLUE.

DECODE ATTRIBUTES



LENGTH 56 bytes

Using the routine

POKE the line number into location 23296.
POKE the column number into location 23297.
The attribute values are returned in the following locations:
FLASH in 23298
BRIGHT in 23299
PAPER in 23300
INK in 23301

Decode attributes data

33, 0, 91, 126, 35, 94, 254, 22, 48, 54,
33, 0, 88, 22, 0, 25, 17, 32, 0, 254,
0, 40, 4, 25, 61, 24, 248, 17, 2, 91,
126, 7, 230, 1, 18, 19, 126, 7, 7, 230,
1, 18, 19, 126, 15, 15, 15, 230, 7, 18,
19, 126, 230, 7, 18, 201

Example 1.6

```
 10 REM DECODE ATTRIBUTES
 20 REM set colours
 30 PAPER 7: INK 0: BORDER 7
 40 CLS
100 REM set new ramtop
110 CLEAR 65311: REM 16K=32543
120 LET start=65312: REM 16K=32
544
200 REM place routine
210 LET length=56
220 LET s=start
230 FOR i=1 TO length
240 READ n
250 POKE s,n
260 LET s=s+1
270 NEXT i
300 REM random attributes
310 LET line=0
320 LET column=16
330 PRINT AT line,column;"A"
340 RANDOMIZE 0
350 POKE 22544,INT (RND*256)
400 REM set x,y
410 POKE 23296,line
420 POKE 23297,column
430 RANDOMIZE USR start
500 REM print attributes
510 PRINT AT 10,0;"Flash  = ";P
EEK 23298
520 PRINT "Bright = ";PEEK 2329
9
530 PRINT "Paper  = ";PEEK 2330
0
540 PRINT "Ink    = ";PEEK 2330
1
550 PRINT AT 21,0;"Press a to t
ry again"
560 LET k$=INKEY$
570 IF k$="" THEN  GO TO 560
580 IF k$="a" THEN  GO TO 340
1000 REM DECODE ATTRIBUTES
```

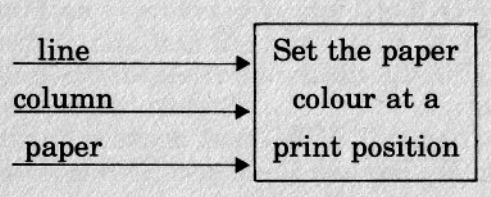
Program 1.6 *(continues)*

```
1010 DATA 33,0,91,126,35,94,254,
22,48,54
1020 DATA 33,0,88,22,0,25,17,32,
0,254
1030 DATA 0,40,4,25,61,24,248,17
,2,91
1040 DATA 126,7,230,1,18,19,126,
7,7,230
1050 DATA 1,18,19,126,15,15,15,2
30,7,18
1060 DATA 19,126,230,7,18,201
```

Program 1.6

SET PAPER may be used to set the paper colour at any print position, regardless of the current paper colour.

SET PAPER



LENGTH 43 bytes

Using the routine

POKE the line number into location 23296.
POKE the column number into location 23297.
POKE the PAPER number into location 23298.

Set paper data

33, 0, 91, 126, 35, 94, 254, 22, 48, 32,
33, 0, 88, 22, 0, 25, 17, 32, 0, 254,
0, 40, 4, 25, 61, 24, 248, 235, 33, 2,
91, 26, 15, 15, 15, 230, 248, 174, 7, 7,
7, 18, 201

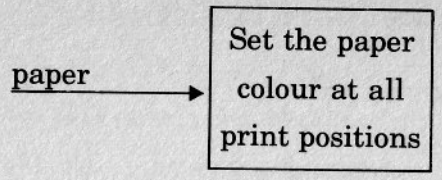
Example 1.7

```
  10 REM SET PAPER COLOUR
  15 REM AT PRINT POSITION
  20 REM set colours
  30 PAPER 7: INK 0: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65324: REM 16K=32556
 120 LET start=65325: REM 16K=32
557
 200 REM place routine
 210 LET length=43
 220 LET s=start
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 300 REM fetch paper colour
 310 LET line=10
 320 PRINT AT line,2;"ENTER THE
PAPER COLOUR (0-7)"
 330 INPUT paper
 340 IF paper>7 THEN  GO TO 500
 400 REM set paper colour
 410 POKE 23296,line
 420 POKE 23298,paper
 430 FOR c=2 TO 29
 440 POKE 23297,c
 450 RANDOMIZE USR start
 460 NEXT c
 500 REM finished
 510 PAPER 7: INK 0
1000 REM SET PAPER
1010 DATA 33,0,91,126,35,94,254,
22,48,32
1020 DATA 33,0,88,22,0,25,17,32,
0,254
1030 DATA 0,40,4,25,61,24,248,23
5,33,2
1040 DATA 91,26,15,15,15,230,248
,174,7,7
1050 DATA 7,18,201
```

Program 1.7

SET ALL PAPER may be used to set or change the paper colour at all the print positions on the screen, regardless of the current value of PAPER. Like the other attribute routines, this routine does not change the current BASIC value of any attribute; it merely acts on the attribute file directly.

SET ALL PAPER



LENGTH 28 bytes

Using the routine

POKE the PAPER number into location 23296.

Set all paper data

33, 0, 91, 17, 0, 88, 14, 11, 6, 64,
26, 15, 15, 230, 248, 174, 7, 7, 7,
18, 19, 16, 242, 13, 32, 237, 201

Example 1.8

If some of the ink is rather difficult to see against the paper colours the intended message can be seen by looking at the listing. Changing to a more suitable paper colour will reveal the message!

```
 10 REM CHANGE ALL THE PAPER
 20 REM set colours
 30 PAPER 7: INK 0: BORDER 7
 40 CLS
100 REM set new ramtop
110 CLEAR 65339: REM 16K=32571
120 LET start=65340: REM 16K=32
572
200 REM place routine
210 LET length=28
220 LET s=start
230 FOR i=1 TO length
240 READ n
250 POKE s,n
260 LET s=s+1
270 NEXT i
300 REM print message
310 PAPER 1: PRINT AT 5,13;"ENT
ER"
320 PAPER 2: PRINT AT 7,14;"NEW
"
330 PAPER 3: PRINT AT 9,13;"PAP
ER"
340 PAPER 4: PRINT AT 11,13;"CO
LOUR"
```

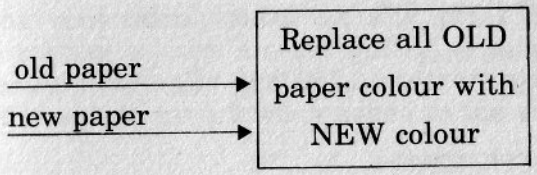


```
350 PAPER 5: PRINT AT 13,14;"NO
W"
360 PAPER 6: PRINT AT 15,14;"AN
D"
370 PAPER 7: PRINT AT 17,13;"WA
TCH"
380 INPUT ink
390 IF ink>7 THEN  GO TO 700
400 REM change ink colour
410 POKE 23296,ink
420 RANDOMIZE USR start
700 REM finished
710 PAPER 7: INK 0
1000 REM SET ALL PAPER
1010 DATA 33,0,91,17,0,88,14,11,
6,64
1020 DATA 26,15,15,15,230,248,17
4,7,7,7
1030 DATA 18,19,16,242,13,32,237
,201
```

Program 1.8

Although the SET ALL PAPER routine offers the facility for changing the colour of all the paper on the screen, this may be taking matters too far. In some instances, what is needed is the facility for changing all occurrences of a specific paper colour to another, specified, colour. This can be done by using the SWAP PAPER routine. Specify the paper colour which is to be replaced, and the new paper colour which is to be used in its place; the routine then searches the attribute file for occurrences of the specified paper colour, and replaces that colour with the chosen new colour at every occurrence.

SWAP PAPER



LENGTH 39 bytes

Using the routine

POKE the OLD paper colour into location 23296.
POKE the NEW paper colour into location 23297.

Swap paper data

33, 0, 91, 17, 0, 88, 14, 11, 6, 64,
26, 15, 15, 15, 230, 7, 190, 32, 13, 26,
15, 15, 15, 230, 248, 35, 174, 43, 7, 7,
7, 18, 19, 16, 231, 13, 32, 226, 201

Example 1.9

```
  10 REM SWAP PAPER
  20 REM set colours
  30 PAPER 7: INK 0: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65328: REM 16K=32560
 120 LET start=65329: REM 16K=32
561
 200 REM place routine
 210 LET length=39
 220 LET s=start
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 300 REM create colours
 310 PRINT : PRINT : PRINT
 320 FOR p=0 TO 7
 330 PAPER p
 340 PRINT TAB (31);" ": PRINT T
AB (31);" "
 350 NEXT p
 400 REM change colour
 410 INPUT "Change which colour
(0-7)";c
 420 IF c>7 THEN  GO TO 700
 430 INPUT "Replacement colour (
0-7)";r
 440 IF r>7 THEN  GO TO 700
 450 REM set up colours
 460 POKE 23296,c
 470 POKE 23297,r
 480 REM execute colour change
 490 RANDOMIZE USR start
 500 IF c<8 THEN  GO TO 410
 700 REM finished
 710 PAPER 7: INK 0
1000 REM SWAP PAPER
1010 DATA 33,0,91,17,0,88,14,11,
6,64
1020 DATA 26,15,15,15,230,7,190,
32,13,26
```



```
1030 DATA 15,15,15,230,248,35,17
4,43,7,7
1040 DATA 7,18,19,16,231,13,32,2
26,201
```

Program 1.9

The paper colours are shown using line 340 which simply TABs across print positions. This has the effect of activating the display of paper colours, in BASIC.

In BASIC, the number 9 may be used with the keywords PAPER and INK, to denote contrasting colours. Contrasting in this case means:

PAPER – use PAPER 7 if INK = 0, 1, 2, or 3
– use PAPER 0 if INK = 4, 5, 6, or 7
INK – use INK 7 if PAPER = 0, 1, 2, or 3
– use INK 0 if PAPER = 4, 5, 6, or 7

As usual, the use of either PAPER 9 or INK 9 requires a reference to a print position, implying that a character will be printed there. While it is possible in many instances to examine the contents of the print position, and to determine its contents (using SCREEN$), it is not always convenient. In BASIC, the method would be:

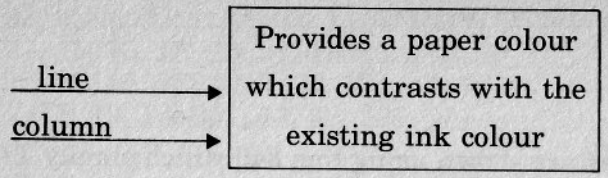
1. Examine the print position, identifying its contents using SCREEN$.
2. Re-print the character at that print position, using PAPER or INK 9.

Unfortunately, the SCREEN$ function is limited to identifying only characters, and only those with codes in the range 32–127, so it is rather limited in practice.

However, there are routines which can be used to help overcome these limitations. CONTRASTING PAPER provides the facility for using contrasting paper at any print position. The rules for contrasting colours are not quite the same as those used by the Spectrum normally. Instead, a contrasting colour is defined as one which adds to give 7 when added to the existing ink colour.
So,

BLUE paper contrasts with YELLOW ink, since 1 + 6 = 7

CONTRASTING PAPER



LENGTH 48 bytes

Using the routine

POKE the line number into location 23296.
POKE the column number into location 23297.

Contrasting paper data

33, 0, 91, 126, 35, 94, 254, 22, 48, 37,
33, 0, 88, 22, 0, 25, 17, 32, 0, 254,
0, 40, 4, 25, 61, 24, 248, 235, 26, 230,
7, 103, 62, 7, 148, 103, 26, 15, 15, 15,
230, 248, 180, 7, 7, 7, 18, 201

Example 1.10

Try substituting various values of INK at line 320, to see the contrasting paper colours.

```
  10 REM USE CONTRASTING PAPER
  15 REM AT PRINT POSITION
  20 REM set colours
  30 PAPER 7: INK 0: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65319: REM 16K=32551
 120 LET start=65320: REM 16K=32
552
 200 REM place routine
 210 LET length=48
 220 LET s=start
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 300 REM fetch paper colour
 310 LET line=10
 320 INK 6
 330 PRINT AT line,2;"PRESS c FO
R CONTRASTING PAPER"
```

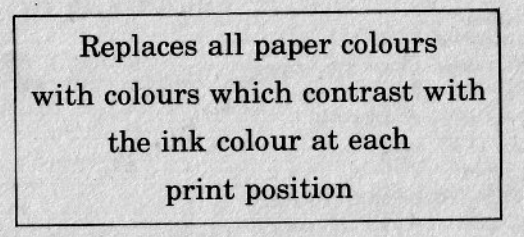


```
 340 LET k$=INKEY$
 350 IF k$="" THEN  GO TO 340
 360 IF k$<>"c" THEN  GO TO 500
 400 REM use contrasting paper
 410 POKE 23296,line
 420 FOR p=2 TO 30
 430 POKE 23297,p
 440 RANDOMIZE USR start
 450 NEXT p
 500 REM finished
 510 PAPER 7: INK 0
1000 REM CONTRASTING PAPER
1010 DATA 33,0,91,126,35,94,254,
22,48,37
1020 DATA 33,0,88,22,0,25,17,32,
0,254
1030 DATA 0,40,4,25,61,24,248,23
5,26,230
1040 DATA 7,103,62,7,148,103,26,
15,15,15
1050 DATA 230,248,180,7,7,7,18,2
01
```

Program 1.10

The same process may be carried out for the whole attribute file, using the routine entitled ALL CONTRASTING PAPER.

ALL CONTRASTING PAPER



LENGTH 33 bytes

Using the routine

No values need to be set before using this routine, since it deals with the whole screen and can examine the existing ink colours.

All contrasting paper data

17, 0, 88, 14, 11, 6, 64, 26, 230, 7,
103, 62, 7, 148, 103, 26, 15, 15, 15, 230,
248, 180, 7, 7, 7, 18, 19, 16, 234, 13,
32, 229, 201

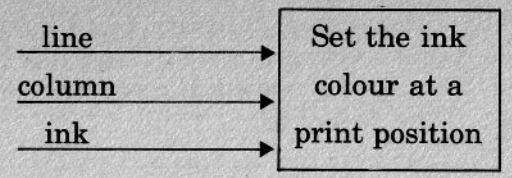
Example 1.11

The ink colour is initially set randomly, before printing the message in this example, so it may be unreadable at first if it has been set to the same colour as the paper. However, the purpose of the routine is to ensure that it will be readable!

```
  10 REM USE ALL CONTRASTING PAP
ER
  20 REM set colours
  30 PAPER 7: INK 0: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65334: REM 16K=32566
 120 LET start=65335: REM 16K=32
567
 200 REM place routine
 210 LET length=33
 220 LET s=start
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 300 REM print message
 310 REM random ink
 320 RANDOMIZE 0
 330 LET ink=INT (RND*8)
 340 INK ink
 350 PRINT AT 10,0;"Press c for
contrasting paper"
 360 LET k$=INKEY$
 370 IF k$="" THEN  GO TO 360
 380 IF k$<>"c" THEN  GO TO 700
 400 REM contrasting PAPER
 410 RANDOMIZE USR start
 700 REM finished
1000 REM ALL CONTRASTING PAPER
1010 DATA 17,0,88,14,11,6,64,26,
230,7
1020 DATA 103,62,7,148,103,26,15
,15,15,230
1030 DATA 248,180,7,7,7,18,19,16
,234,13
1040 DATA 32,229,201
```

Program 1.11

Exactly the same sorts of routines are available for manipulating the ink colours as for the paper colours.



LENGTH 37 bytes

Using the routine

POKE the line number into location 23296.
POKE the column number into location 23297.
POKE the INK number into location 23298.

Set ink data

33, 0, 91, 126, 35, 94, 254, 22, 48, 26,
33, 0, 88, 22, 0, 25, 17, 32, 0, 254,
0, 40, 4, 25, 61, 24, 248, 235, 33, 2,
91, 26, 230, 248, 174, 18, 201

Example 1.12

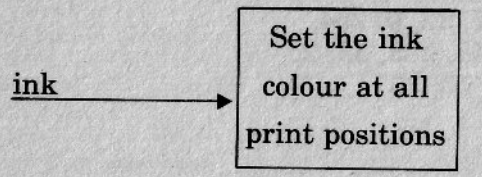
```
  10 REM SET INK COLOUR
  15 REM AT PRINT POSITION
  20 REM set colours
  30 PAPER 7: INK 0: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65330: REM 16K=32569
 120 LET start=65331: REM 16K=32
570
 200 REM place routine
 210 LET length=37
 220 LET s=start
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 300 REM fetch ink colour
 310 LET line=10
 320 PRINT AT line,4;"ENTER THE
INK COLOUR (0-7)"
 330 INPUT ink
 340 IF ink>7 THEN  GO TO 500
 400 REM set ink colour
 410 POKE 23296,line
 420 POKE 23298,ink
```

Program 1.12 *(continues)*

```
 430 FOR c=4 TO 29
 440 POKE 23297,c
 450 RANDOMIZE USR start
 460 NEXT c
 500 REM finished
 510 PAPER 7: INK 0
1000 REM SET INK
1010 DATA 33,0,91,126,35,94,254,
22,48,26
1020 DATA 33,0,88,22,0,25,17,32,
0,254
1030 DATA 0,40,4,25,61,24,248,23
5,33,2
1040 DATA 91,26,230,248,174,18,2
01
```

Program 1.12

SET ALL INK



LENGTH 22 bytes

Using the routine

POKE the INK number into location 23296.

Set all ink data

33, 0, 91, 17, 0, 88, 14, 11, 6, 64,
26, 230, 248, 174, 18, 19, 16, 248, 13, 32,
243, 201

Example 1.13

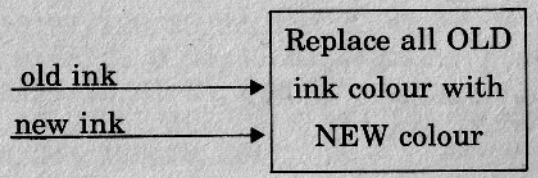
```
 10 REM CHANGE ALL INK
 20 REM set colours
 30 PAPER 7: INK 0: BORDER 7
 40 CLS
100 REM set new ramtop
110 CLEAR 65345: REM 16K=32577
120 LET start=65346: REM 16K=32
578
200 REM place routine
210 LET length=22
220 LET s=start
```



```
230 FOR i=1 TO length
240 READ n
250 POKE s,n
260 LET s=s+1
270 NEXT i
300 REM print message
310 INK 1: PRINT AT 5,13;"ENTER
"
320 INK 2: PRINT AT 7,14;"NEW"
330 INK 3: PRINT AT 9,14;"INK"
340 INK 4: PRINT AT 11,13;"COLO
UR"
350 INK 5: PRINT AT 13,14;"NOW"
360 INK 6: PRINT AT 15,14;"AND"
370 INK 0: PRINT AT 17,13;"WATC
H"
380 INPUT ink
390 IF ink>7 THEN  GO TO 700
400 REM change ink colour
410 POKE 23296,ink
420 RANDOMIZE USR start
700 REM finished
1000 REM SET ALL INK
1010 DATA 33,0,91,17,0,88,14,11,
6,64
1020 DATA 26,230,248,174,18,19,1
6,248,13,32
1030 DATA 243,201
```

Program 1.13

SWAP INK



LENGTH 30 bytes

Using the routine

POKE the OLD ink colour into location 23296.
POKE the NEW ink colour into location 23297.

Swap ink data

33, 0, 91, 17, 0, 88, 14, 11, 6, 64,
26, 230, 7, 190, 32, 7, 26, 230, 248, 35,
174, 43, 18, 19, 16, 240, 13, 32, 235, 201

Example 1.14

```
  10 REM SWAP INK
  20 REM set colours
  30 PAPER 7: INK 0: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65337: REM 16K=32569
 120 LET start=65338: REM 16K=32
570
 150 REM define bar symbol
 160 LET b$=""
 170 FOR i=1 TO 10
 180 LET b$=b$+CHR$ 143
 190 NEXT i
 200 REM place routine
 210 LET length=30
 220 LET s=start
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 300 REM create colour bars
 310 FOR b=0 TO 7
 320 INK b
 330 PRINT AT 3+b*2,10;b$
 340 NEXT b
 400 REM change colour
 410 INPUT "Change which colour
(0-7)";c
 420 IF c>7 THEN  GO TO 700
 430 INPUT "Replacement colour (
0-7)";r
 440 IF r>7 THEN  GO TO 700
 450 REM set up colours
 460 POKE 23296,c
 470 POKE 23297,r
 480 REM execute colour change
 490 RANDOMIZE USR start
 500 IF c<8 THEN  GO TO 410
 700 REM finished
 710 PAPER 7: INK 0
1000 REM SWAP INK
1010 DATA 33,0,91,17,0,88,14,11,
6,64
1020 DATA 26,230,7,190,32,7,26,2
30,248,35
1030 DATA 174,43,18,19,16,240,13
,32,235,201
```

Program 1.14

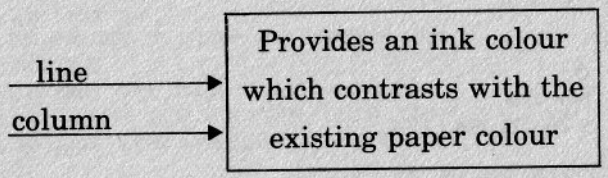
The solid bars of colour are created using CHR$143 (graphics character 8). Notice the effect as a colour bar is changed to the same

colour as one of the existing bars; when that colour is changed, both bars change (of course). This shows that some care is required when using the routine to swap the colours of two sets of ink on the screen. An easy way to do this is to use an intermediate colour; for example:

Changing BLUE to YELLOW ink.
First find an ink colour which is not in use on the screen.
Change the BLUE to the unused colour;
change the YELLOW to BLUE; and then
change the unused colour to YELLOW.

The same rules for contrasting ink colours are used as for contrasting paper; the ink must add to the existing paper to give 7.

CONTRASTING INK



LENGTH 45 bytes

Using the routine

POKE the line number into location 23296.
POKE the column number into location 23297.

Contrasting ink data

33, 0, 91, 126, 35, 94, 254, 22, 48, 34,
33, 0, 88, 22, 0, 25, 17, 32, 0, 254,
0, 40, 4, 25, 61, 24, 248, 235, 26, 15,
15, 15, 230, 7, 103, 62, 7, 148, 103, 26,
230, 248, 180, 18, 201

Example 1.15

```
  10 REM USE CONTRASTING INK
  15 REM AT PRINT POSITION
  20 REM set colours
  30 PAPER 7: INK 0: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65322: REM 16K=32554
 120 LET start=65323: REM 16K=32
555
 200 REM place routine
 210 LET length=45
```

Program 1.15 *(continues)*

```
 220 LET s=start
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 300 REM fetch paper colour
 310 LET line=10
 320 PAPER 6: INK 7
 330 PRINT AT line,4;"PRESS c FO
R CONTRASTING INK"
 340 LET k$=INKEY$
 350 IF k$="" THEN  GO TO 340
 360 IF k$<>"c" THEN  GO TO 500
 400 REM use contrasting ink
 410 POKE 23296,line
 420 FOR p=2 TO 30
 430 POKE 23297,p
 440 RANDOMIZE USR start
 450 NEXT p
 500 REM finished
 510 PAPER 7: INK 0
1000 REM CONTRASTING INK
1010 DATA 33,0,91,126,35,94,254,
22,48,34
1020 DATA 33,0,88,22,0,25,17,32,
0,254
1030 DATA 0,40,4,25,61,24,248,23
5,26,15
1040 DATA 15,15,230,7,103,62,7,1
48,103,26
1050 DATA 230,248,180,18,201
```

Program 1.15

ALL CONTRASTING INK

Replaces all ink colours with colours which contrast with the paper colour at each print position

LENGTH 30 bytes

Using the routine

No values need be set before using this routine.

All contrasting ink data

17, 0, 88, 14, 11, 6, 64, 26, 15, 15,
15, 230, 7, 103, 62, 7, 148, 103, 26, 230,
248, 180, 18, 19, 16, 237, 13, 32, 232, 201

Example 1.16

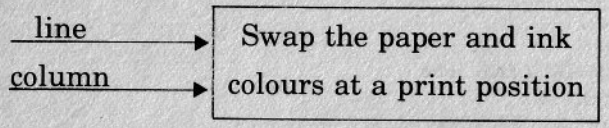
The paper colour is randomly set in line 330, so it is possible that the message may be invisible. However, pressing 'c' ensures that this will change.

```
  10 REM USE CONTRASTING INK
  20 REM set colours
  30 PAPER 7: INK 0: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65337: REM 16K=32569
 120 LET start=65338: REM 16K=32
570
 200 REM place routine
 210 LET length=30
 220 LET s=start
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 300 REM print message
 310 REM random paper
 320 RANDOMIZE 0
 330 LET paper=INT (RND*8)
 340 PAPER paper
 350 PRINT AT 10,0;"Press c for
contrasting INK"
 360 LET k$=INKEY$
 370 IF k$="" THEN  GO TO 360
 380 IF k$<>"c" THEN  GO TO 700
 400 REM contrasting INK
 410 RANDOMIZE USR start
 700 REM finished
1000 REM CONTRASTING INK
1010 DATA 17,0,88,14,11,6,64,26,
15,15
1020 DATA 15,230,7,103,62,7,148,
103,26,230
1030 DATA 248,180,18,19,16,237,1
3,32,232,201
```

Program 1.16

An inversion facility can be useful for highlighting areas on the screen – inversion in this case meaning swapping paper for ink and ink for paper. FLASH provides the means of swapping paper and ink in BASIC, but it is a repeated effect, once chosen. Simply swapping the colours is a one-off operation which produces a rather different effect. As before, this effect is accomplished by acting directly on the attribute file and does not depend on printing something on the screen, nor does it alter the values of PAPER and INK in BASIC.

SWAP PAPER AND INK



LENGTH 54 bytes

Using the routine

POKE the line number into location 23296.
POKE the column number into location 23297.

Swap paper and ink data

33, 0, 91, 126, 35, 94, 254, 22, 48, 43,
33, 0, 88, 22, 0, 25, 17, 32, 0, 254,
0, 40, 4, 25, 61, 24, 248, 235, 26, 230,
7, 111, 26, 15, 15, 15, 230, 7, 103, 26,
15, 15, 15, 230, 248, 181, 7, 7, 7, 230,
248, 180, 18, 201

Example 1.17

```
 10 REM SWAP PAPER AND INK
 15 REM AT PRINT POSITION
 20 REM set colours
 30 PAPER 7: INK 0: BORDER 7
 40 CLS
100 REM set new ramtop
110 CLEAR 65313: REM 16K=32545
120 LET start=65314: REM 16K=32
546
200 REM place routine
210 LET length=54
220 LET s=start
230 FOR i=1 TO length
240 READ n
250 POKE s,n
260 LET s=s+1
270 NEXT i
300 REM message
310 PAPER 2: INK 6: PRINT AT 10
,0;"PRESS c TO SWAP PAPER + INK"
320 LET line=10
330 LET column=6: REM letter c
340 LET k$=INKEY$
350 IF k$="" THEN  GO TO 340
360 IF k$<>"c" THEN  GO TO 500
400 REM swap paper + ink
410 POKE 23296,line
420 POKE 23297,column
430 RANDOMIZE USR start
500 REM finished
510 PAPER 7: INK 0
1000 REM SWAP PAPER AND INK
1010 DATA 33,0,91,126,35,94,254,
22,48,43
1020 DATA 33,0,88,22,0,25,17,32,
0,254
1030 DATA 0,40,4,25,61,24,248,23
5,26,230
1040 DATA 7,111,26,15,15,15,230,
7,103,26
1050 DATA 15,15,15,230,248,181,7
,7,7,230
1060 DATA 248,180,18,201
```

Program 1.17

Try changing the values of paper and ink which are set in line 310.

SWAP ALL PAPER AND INK

Swap the paper and ink colours at all print positions

LENGTH 39 bytes

Using the routine

No values need be set before using this routine.

Swap all paper and ink data

17, 0, 88, 14, 11, 6, 64, 26, 230, 7,
111, 26, 15, 15, 15, 230, 7, 103, 26, 15,
15, 15, 230, 248, 181, 7, 7, 7, 230, 248,
180, 18, 19, 16, 228, 13, 32, 223, 201

Example 1.18

```
  10 REM SWAP ALL PAPER AND INK
  20 REM set colours
  30 PAPER 7: INK 0: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65328: REM 16K=32560
 120 LET start=65329: REM 16K=32
561
 200 REM place routine
 210 LET length=39
 220 LET s=start
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 300 REM print message
 310 PRINT AT 10,0;"Press s to s
wap paper and ink"
 320 LET k$=INKEY$
 330 IF k$="" THEN  GO TO 320
 340 IF k$<>"s" THEN  GO TO 700
 400 REM swap paper and ink
 410 RANDOMIZE USR start
 700 REM finished
1000 REM SWAP ALL PAPER AND INK
1010 DATA 17,0,88,14,11,6,64,26,
230,7
1020 DATA 111,26,15,15,15,230,7,
103,26,15
1030 DATA 15,15,230,248,181,7,7,
7,230,248
1040 DATA 180,18,19,16,228,13,32
,223,201
```

Program 1.18

CLS in line 40 ensures that all the paper on the screen is set to 7. That explains what happens to the 'empty' part of the screen. Place additional statements in the program to print other messages on the screen using different colours of paper and ink, then watch them all change when the routine is executed.

Hide and reveal

The SWAP INK routine provides the basis for an interesting effect in which a message, or any other picture, is flashed on and off the screen.

To hide anything printed or drawn on the screen, change the INK to the same colour as the PAPER.

To reveal hidden pixels, change the INK to any colour which is different from the PAPER.

Example 1.19

In this example, a pair of large letters is produced using graphics characters (graphics character 8).

The letters are repeatedly hidden, then revealed, by changing the INK to the same colour as the PAPER, then changing the INK to a colour which contrasts with the PAPER (INK 6). In fact, any INK colour could be used as long as it is not the same as the PAPER colour (0 in Program 1.19).

```
  10 REM HIDE AND REVEAL
  20 REM set colours
  30 PAPER 0: INK 6: BORDER 0
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65336: REM 16K=32568
 120 LET start=65337: REM 16K=32
569
 130 LET length=30
 140 LET s=start
 150 FOR i=1 TO length
 160 READ n
 170 POKE s,n
 180 LET s=s+1
 190 NEXT i
 200 REM create message
 210 LET a$=CHR$ 143+CHR$ 32+CHR
$ 32+CHR$ 143+CHR$ 32+CHR$ 143+C
HR$ 143+CHR$ 143
 220 LET b$=CHR$ 143+CHR$ 32+CHR
$ 32+CHR$ 143+CHR$ 32+CHR$ 32+CH
R$ 143
 230 LET c$=CHR$ 143+CHR$ 143+CH
R$ 143+CHR$ 143+CHR$ 32+CHR$ 32+
CHR$ 143
 240 PRINT AT 10,12;a$
 250 PRINT AT 11,12;b$
 260 PRINT AT 12,12;c$
 270 PRINT AT 13,12;b$
 280 PRINT AT 14,12;a$
```

Program 1.19 *(continues)*

```
 300 REM hide and reveal
 310 FOR j=1 TO 20
 320 PAUSE 25
 330 REM hide
 340 POKE 23296,6: REM old INK
 350 POKE 23297,0: REM new INK
 360 RANDOMIZE USR start
 370 PAUSE 25
 380 REM reveal
 390 POKE 23296,0: REM old INK
 400 POKE 23297,6: REM new INK
 410 RANDOMIZE USR start
 420 NEXT j
 500 REM restore colours
 510 PAUSE 25
 520 PAPER 7: INK 0: BORDER 7: C
LS
 600 REM SWAP INK
 610 DATA 33,0,91,17,0,88,14,11,
6,64
 620 DATA 26,230,7,190,32,7,26,2
30,248,35
 630 DATA 174,43,18,19,16,240,13
,32,235,201
```

Program 1.19

Titletype

TITLETYPE is a short program (1.20) which can be used to print messages on the screen, using large characters.

Lines 300–410 read strings which are to be printed, from DATA at the end of the program, and print each character in the string individually. The scale at which each string is to be printed can be specified as can the starting positions of the strings. Lines 1010 and 1020 show how the data should be presented.

The first item of data should be the string to be plotted. Following this string should be:

x co-ordinate of the bottom left corner of the first character.
y co-ordinate of the bottom left corner of the first character.
x scale.
y scale.
Rotation.

These values are read in line 330, and poked into the appropriate locations using line 340, in preparation for plotting.

For each character in the string (lines 350–400), the *x* co-ordinate of the bottom left corner is set (line 360) and the character code is determined (line 370). The ROTATED SCALED CHARACTER PLOTTER routine is then used (line 380) to plot the character.

The ideas and techniques presented in this short program can be extended to provide the basis for a sophisticated message-generating program which allows the keyboard to be used as a very fancy typewriter.

Run Program 1.20, and save the resulting display, using

SAVE "title" SCREEN$

This display will be used later, at the end of Chapter 7, to illustrate some of the routines given there.

```
  10 REM TITLETYPE
  20 REM set colours
  30 PAPER 6: INK 1: BORDER 6
  40 CLS
 100 REM set new ramtop
 110 CLEAR 64953: REM 16K=32185
 120 LET start=64954: REM 16K=32
186
 200 REM place routine
 210 LET length=414
 220 LET s=start
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 300 REM fetch and print message
s
 310 READ m: REM no. of strings
 320 FOR i=1 TO m
 330 READ a$,x,y,xs,ys,r
 340 POKE 23297,y: POKE 23299,xs
: POKE 23300,ys: POKE 23301,r
 350 FOR j=1 TO LEN a$
 360 POKE 23296,x
 370 POKE 23298,CODE a$(j)
 380 RANDOMIZE USR start
 390 LET x=x+8*xs
 400 NEXT j
 410 NEXT i
 600 REM ROTATED SCALED CHARACTE
R PLOTTER
 610  DATA 58,5,91,254,4,48,43,5
8,2,91,254,32,56,36,254,128,56,8
,254,144
 620 DATA 56,28,254,165,56,7,33,
7,61,214,32,24,6,42,123,92,43,21
4,143,17
 630 DATA 8,0,254,0,40,6,25,61,2
4,248,24,104,58,5,91,254,0,32,10
,17
 640 DATA 17,91,1,8,0,237,184,24
,82,254,2,32,25,6,8,17,10,91,197
,26
```

Program 1.20 *(continues)*

```
 650 DATA 79,126,6,8,23,203,25,1
6,251,121,18,193,19,43,16,238,24
,53,254,3
 660 DATA 32,25,6,8,17,17,91,126
,197,6,8,23,235,78,203,25,113,23
5,27,16
 670 DATA 246,193,43,16,235,24,2
4,6,8,17,17,91,126,197,6,8,203,3
1,235,78
 680 DATA 203,17,113,235,27,16,2
45,193,43,16,234,33,17,91,229,24
,2,24,70,33
 690 DATA 0,91,126,79,35,126,71,
33,4,91,126,35,35,35,119,35,35,6
2,0,119
 700 DATA 225,24,23,241,193,225,
43,229,33,4,91,126,35,35,35,119,
225,24,3,241
 710 DATA 193,225,58,0,91,79,126
,229,197,245,33,8,91,62,0,119,24
1,7,245,33
 720 DATA 3,91,126,35,35,35,119,
24,8,24,103,24,206,24,220,24,235
,62,16,128
 730 DATA 71,33,0,64,254,128,48,
9,17,0,8,25,254,64,48,1,25,203,1
84,203
 740 DATA 176,62,63,144,17,32,0,
254,8,56,5,214,8,25,24,247,254,0
,40,4
 750 DATA 61,36,24,248,121,254,8
,56,5,214,8,35,24,247,79,62,7,14
5,55,63
 760 DATA 87,71,94,254,0,40,4,20
3,11,16,252,241,245,56,4,203,131
,24,2,203
 770 DATA 195,122,66,254,0,40,4,
203,3,16,252,115,24,8,24,77,24,1
49,24,149
 780 DATA 24,149,33,6,91,126,61,
119,254,0,40,7,241,193,12,197,24
5,24,134,33
 790 DATA 8,91,126,60,119,254,8,
40,7,241,193,12,197,245,24,220,3
3,7,91,126
 800 DATA 61,119,254,0,40,7,241,
193,4,197,245,24,201,33,9,91,126
,60,119,254
 810 DATA 8,40,7,241,193,4,197,2
45,24,182,241,193,225,201
1000 REM MESSAGES
1005 DATA 2
1010 DATA "SPECTRUM",0,104,4,4,0
1020 DATA "GRAPHICS",32,48,3,3,0
```

Program 1.20

2 DRAWING

Pixels

While the PRINT command simply copies a pre-defined pattern into an 8 byte section of the display file, the PLOT command can turn individual pixels (or bits) ON. This provides an accurate means of constructing drawings on the screen, since there are

32 bytes × 8 bits = 256 pixel positions across the screen, and 192 pixel positions from top to bottom.

As with the PRINT command, there is an area 16 pixels high at the bottom of the screen which is reserved for input, so the effective size of the plotting area is 256 × 176 pixels.

Unfortunately, the PLOT command suffers from a rather surprising disadvantage – it cannot turn pixels OFF! A pixel can be made 'invisible' by setting INK to the same colour as the PAPER, but it cannot be truly turned off, i.e., the corresponding bit cannot be reset to 0 by this command.

A second disadvantage of PLOT is revealed by considering the attribute file. The attributes of each PRINT position are held in a single byte. This means that all the pixels inside an 8 byte PRINT position must have the same INK and PAPER colours. This can cause problems. Plotting a pixel may change the colour of other, adjacent, pixels which were originally plotted in a different INK colour. Some invisible pixels may suddenly appear!

Example 2.1

In this example (Program 2.1), three pixels are plotted. The first (in line 150), is plotted invisibly, since the INK and PAPER are set to the same colour at the time of plotting. The second pixel is visible when first plotted, and is coloured black. This plotting action sets the INK in the surrounding PRINT position black.

Both of these pixels are in the same PRINT position, so when the INK is set to black by the second pixel, the first pixel suddenly becomes visible – very handy if that was the desired effect, but rather annoying otherwise.

The third pixel is also plotted inside the same PRINT position as the first two, but the INK is set to yellow before plotting, so the INK of the first two changes to yellow too. The net result is three visible pixels, all yellow.

```
 10 REM VISIBLE AND INVISIBLE
 20 REM set colours
 30 PAPER 7: INK 7: BORDER 7
 40 CLS
100 REM plot first pixel
110 PRINT  INK 0;AT 21,0;"Press
a key to plot pixel 1"
120 LET k$=INKEY$
130 IF k$="" THEN  GO TO 120
140 PRINT AT 21,0;"
                "
150 PLOT 119,80
200 REM plot second pixel
210 INK 0
220 PRINT AT 21,0;"Press a key
to plot pixel 2"
230 LET k$=INKEY$
240 IF k$="" THEN  GO TO 230
250 PRINT AT 21,0;"
                "
260 PLOT 112,87
300 REM plot third pixel
310 INK 6
320 PRINT  INK 0;AT 21,0;"Press
a key to plot pixel 3"
330 LET k$=INKEY$
340 IF k$="" THEN  GO TO 330
350 PRINT AT 21,0;"
                "
360 PLOT 115,84
400 REM end
410 INK 0
```

Program 2.1

Sometimes, the attributes can be set up in advance, and any pixels which are then plotted will appear in a known colour.

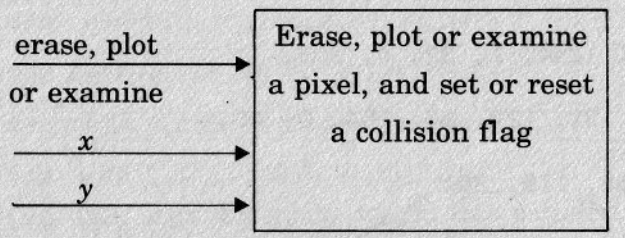
Setting PAPER and INK before clearing the screen is one technique which provides uniform colours over the whole area. A rather more complicated set of colours can be created using some of the attribute setting routines already explained in Chapter 1.

However, some basic problems remain; like truly unplotting pixels by setting the corresponding bit in the display file to 0. This can be accomplished using the PIXEL HANDLER routine which follows. This routine offers the facility for:

1. Plotting a pixel by setting the corresponding bit to 1. The colours used are those which already exist in the PRINT position inside which the pixel is being plotted.
2. Unplotting a pixel, by setting the corresponding bit to 0. This effectively removes the pixel from the screen, irrespective of any subsequent colour changes inside the surrounding PRINT position.
3. A collision flag is set to 1 if the pixel being plotted was placed over a pixel which was already ON. This flag is set irrespective of whether an existing pixel was visible, or not. The routine examines the display file, looking at the bit which corresponds to the pixel about to be plotted. If that bit is already set to 1, the collision flag is set to 1. If the bit is set to 0 before plotting takes place, the collision flag is set to 0.
4. Examination of a pixel may take place without either plotting or unplotting. This process sets the collision flag to 0 if the pixel is OFF, or 1 if it is ON, irrespective of whether the pixel is visible or not.

Note that the BASIC command POINT does not perform this function in the same way. POINT returns 1 if a pixel is INK colour, or 0 if it is PAPER colour. POINT does not identify whether a pixel has been turned ON or OFF; e.g., POINT returns 0 for a pixel which has been plotted in the PAPER colour, as well as for a pixel which has the PAPER colour but has not been plotted (a 'background' pixel).

PIXEL HANDLER



LENGTH 127 bytes

Using the routine

POKE 0, 1, or 2 into location 23301
- ERASE = 0
- PLOT = 1
- EXAMINE = 2

POKE the *x* co-ordinate into location 23298
POKE the *y* co-ordinate into location 23299

Note: On return to BASIC, after execution, location 23302 holds 0 if the pixel at *x,y* was OFF before any alterations made by the routine.

1 if the pixel at *x,y* was ON before any alterations made by the routine.

Pixel handler data

```
33, 2, 91, 78, 35, 70, 62, 175, 184, 56,

115, 62, 16, 128, 71, 33, 0, 64, 254, 128,

48, 9, 17, 0, 8, 25, 254, 64, 48, 1,

25, 203, 184, 203, 176, 62, 63, 144, 17, 32,

0, 254, 8, 56, 5, 214, 8, 25, 24, 247,

254, 0, 40, 4, 61, 36, 24, 248, 121, 254,

8, 56, 5, 214, 8, 35, 24, 247, 79, 62,

7, 145, 55, 63, 87, 71, 94, 254, 0, 40,

4, 203, 11, 16, 252, 62, 0, 203, 67, 40,

2, 62, 1, 50, 6, 91, 58, 5, 91, 254,

2, 40, 23, 48, 21, 254, 1, 32, 4, 203,

195, 24, 2, 203, 131, 122, 66, 254, 0, 40,

4, 203, 3, 16, 252, 115, 201
```

Example 2.2

```
  10 REM PLOTTING PIXELS
  20 REM set colours
  30 PAPER 7: INK 0: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65240: REM 16k=32472
 120 LET start=65241: REM 16k=32
473
 200 REM load routine
 210 LET length=127
 220 LET s=start
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 300 REM draw horizontal line
 310 PLOT 0,88: DRAW 255,0
 400 REM plot points on vertical
 line
 410 POKE 23301,1
 420 LET x=126
 430 POKE 23298,x: REM x
 440 FOR y=175 TO 0 STEP -1
 450 POKE 23299,y: REM y
 460 RANDOMIZE USR start
 470 REM examine collision flag
 480 IF PEEK 23302=1 THEN  PRINT
 AT 0,0;"Collision at ";PEEK 232
98;",";PEEK 23299
 490 NEXT y
1000 REM PIXEL HANDLER
1010 DATA 33,2,91,78,35,70,62,17
6,184,56
1020 DATA 115,62,16,128,71,33,0,
64,254,128
1030 DATA 48,9,17,0,8,25,254,64,
48,1
1040 DATA 25,203,184,203,176,62,
63,144,17,32
1050 DATA 0,254,8,56,5,214,8,25,
24,247
1060 DATA 254,0,40,4,61,36,24,24
8,121,254
1070 DATA 8,56,5,214,8,35,24,247
,79,62
1080 DATA 7,145,55,63,87,71,94,2
54,0,40
1090 DATA 4,203,11,16,252,62,0,2
03,67,40
1100 DATA 2,62,1,50,6,91,58,5,91
,254
1110 DATA 2,40,23,48,21,254,1,32
,4,203
1120 DATA 195,24,2,203,131,122,6
6,254,0,40
1130 DATA 4,203,3,16,252,115,201
```

Program 2.2

Line 410 places 1 in location 23301. This prepares the routine for plotting. *x* is fixed at 126 by line 420, and the FOR . . . NEXT loop in line 440 indexes the value of *y* from 175 (top of the screen) to 0 (bottom of the screen). Line 450 fixes the value of *y*, and line 460 executes the routine, plotting the pixel. In line 480, the collision flag is examined, and a suitable message is printed if a collision took place. This flag is used to detect the intersection of the existing

horizontal line and the vertical line which is being produced by plotting the individual pixels. Executing the PIXEL PLOTTER routine does not change the values of *x* and *y* in locations 23298 and 23299, so when a collision is detected these locations may be examined to find the co-ordinates of this point.

Instead of running this program with 23301 holding 1, try setting 23301 to 2. The pixels from 126,175 to 126,0 will be scanned and 126,88 will again be identified.

To see the erasing function working, add

320 REM draw vertical line
330 PLOT 126.0: DRAW 0,175

then replace

410 POKE 23301,0

and delete line 480 – or leave it in the program and watch the rather peculiar effect!

The routine will now erase the vertical line instead of drawing it. Note that the pixel at 126,88 is erased, although this may not be apparent at first. To check this, use the following direct commands, immediately after the program has finished running.

POKE 23301,2: POKE 23299,88
RANDOMIZE USR start
PRINT PEEK 23302

The result should be 0, indicating that this pixel is OFF. It may be difficult to see that this pixel is off, but that is because the TV has insufficient resolution to be able to show this clearly.

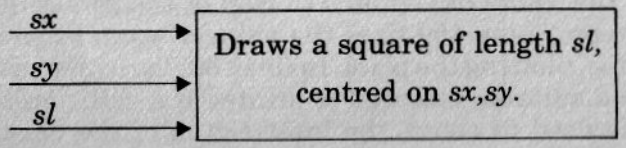
BASIC shapes

PLOT and DRAW are sufficiently powerful commands to allow a lot of impressive graphics to be generated using BASIC. The main drawback is the slow speed of BASIC. However, in many applications this is not a problem.

SQUARES

The following subroutine allows a square to be drawn by specifying its centre, and side length.

SQUARE 1



Using the routine

Set the values of *sx* and *sy* (the *x* and *y* co-ordinates of the centre) and *sl* (the side length) before using the routine.

```
5000 REM SQUARE 1
5010 REM Draws a square
5020 REM of length sl
5030 REM centre (sx,sy)
5040 REM Check it fits on screen
5050 IF sx+sl/2>255 OR sx-sl/2<0
 OR sy+sl/2>175 OR sy-sl/2<0 THE
N  PRINT "square will not fit":
GO TO 5090
5060 PLOT sx-sl/2,sy-sl/2: REM b
ottom left corner
5070 DRAW 0,sl: DRAW sl,0: DRAW
0,-sl: DRAW -sl,0
5080 RETURN
```

Program 2.A

Example 2.3

```
 100 REM TEST SQUARE 1
 110 PAPER 7: INK 0: BORDER 7
 120 CLS
 130 REM Assign start and length
 140 LET sx=128
 150 LET sy=88
 160 FOR i=4 TO 168 STEP 4
 170 LET sl=i
 180 GO SUB 5000
 190 NEXT i
 200 STOP
5000 REM SQUARE 1
5010 REM Draws a square
5020 REM of length sl
5030 REM centre (sx,sy)
5040 REM Check it fits on screen
5050 IF sx+sl/2>255 OR sx-sl/2<0
 OR sy+sl/2>175 OR sy-sl/2<0 THE
N  PRINT "square will not fit":
GO TO 5090
5060 PLOT sx-sl/2,sy-sl/2: REM b
ottom left corner
5070 DRAW 0,sl: DRAW sl,0: DRAW
0,-sl: DRAW -sl,0
5080 RETURN
```

Program 2.3

This program (2.3) uses a loop beginning in line 160 to produce squares of various sizes, centred on the same point.

After running the program, try varying line 160 so that squares of an odd side length are produced, and notice the peculiar effect, for example:

160 FOR i = 5 TO 160 STEP 5

The reason for this is that the routine deals with distances of *sl*/2 from the centre. If *sl* is an odd number, *sl*/2 is rounded by the DRAW command, to a whole number.

Squares produced using SQUARE 1 have their sides parallel to the sides of the screen. The next routine removes this restriction, allowing an angle of tilt, *sa*, to be specified. This angle is measured in radians, clockwise, from the horizontal, as shown in Fig. 2.1.

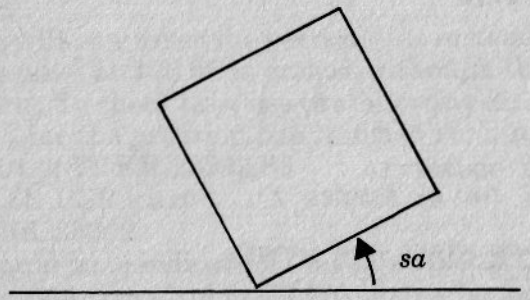
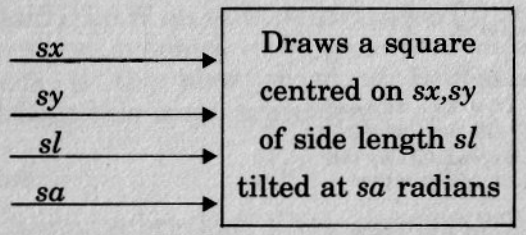


Figure 2.1 A tilted square

SQUARE 2



Using the routine

Assign values to variables *sx* and *sy* (the co-ordinates of the centre), *sl* (the side length) and *sa* (the initial angle of tilt).


```
5100 REM SQUARE 2
5110 REM Draws a square
5120 REM of length sl
5130 REM centre (sx,sy)
5140 REM tilted at sa radians
5200 LET sr=SQR ((sl/2)^2+((sl/2
)^2))
5210 PLOT sx+sr*COS (sa+PI/4),sy
+sr*SIN (sa+PI/4)
5220 DRAW -sl*COS sa,-sl*SIN sa:
 DRAW sl*SIN sa,-sl*COS sa: DRAW
 sl*COS sa,sl*SIN sa: DRAW -sl*S
IN sa,sl*COS sa
5230 RETURN
```

Program 2.B

Example 2.4

```
 100 REM TEST SQUARE 2
 110 PAPER 7: INK 0: BORDER 7
 120 CLS
 130 REM Assign start and length
 140 LET sx=128
 150 LET sy=88
 160 LET sl=50
 170 FOR i=0 TO 2*PI STEP .58
 180 LET sa=i: GO SUB 5100
 190 NEXT i
 200 PAPER 7: INK 0: BORDER 7
 210 STOP
5100 REM SQUARE 2
5110 REM Draws a square
5120 REM of length sl
5130 REM centre (sx,sy)
5140 REM tilted at sa radians
5200 LET sr=SQR ((sl/2)^2+((sl/2
)^2))
5210 PLOT sx+sr*COS (sa+PI/4),sy
+sr*SIN (sa+PI/4)
5220 DRAW -sl*COS sa,-sl*SIN sa:
 DRAW sl*SIN sa,-sl*COS sa: DRAW
 sl*COS sa,sl*SIN sa: DRAW -sl*S
IN sa,sl*COS sa
5230 RETURN
```

Program 2.4

The SQUARE 2 subroutine is called from within a loop beginning at line 170. This loop provides values of *sa* in continually increasing amounts, between 0 and 2*PI radians. Try varying the step size in this loop to produce various effects, for example,

170 FOR i = 0 TO 2*PI STEP 2*PI/8

PLOT and DRAW can be used together to produce line drawings which are formed from a series of lines drawn one after the other. PLOT sets the initial position of a line, and DRAW is used to produce the line. Unless another PLOT is used, the next DRAW command begins drawing where the last line finished. Further PLOT commands are only needed when another series of lines is to be drawn beginning at a new position on the screen.

Example 2.5

```
 100 REM TOFF
 110 REM set colours
 120 PAPER 6: INK 5: BORDER 7
 130 CLS
 200 REM plot and draw picture
 210 PLOT 68,120
 220 DRAW 120,0
 230 PLOT 88,48
 240 DRAW 0,112: DRAW 80,0: DRAW
 0,-112: DRAW -20,-18: DRAW -40,
0: DRAW -20,18
 250 PLOT 100,100
 260 DRAW 18,0
 270 PLOT 138,100
 280 DRAW 18,0
 290 PLOT 119,80
 300 DRAW 5,0
 310 PLOT 132,80
 320 DRAW 5,0
 330 PLOT 108,60
 340 DRAW 20,-10: DRAW 20,10
```

Program 2.5

The basic process of using PLOT and DRAW commands can be refined a little using DATA statements which hold the co-ordinates of the points on the outline of the shape. The first point should be plotted. Subsequent points should be used to calculate the *x* and *y* components of a DRAW command which will draw a linking line to that point from the last plotted point.

Example 2.6

Here *x* and *y* represent the co-ordinates of the last plotted point, while *xn* and *yn* are the co-ordinates of the new point which is to be joined to the last point by a straight line. Reading a pair of negative co-ordinates signals the end of the data.


```
 100 REM DOT TO DOT
 110 REM set colours
 120 PAPER 7: INK 3: BORDER 7
 200 REM join points
 210 READ x,y
 220 PLOT x,y
 230 READ xn,yn
 240 IF xn<0 OR yn<0 THEN  GO TO
 300
 250 DRAW xn-x,yn-y
 260 LET x=xn
 270 LET y=yn
 280 IF x>=0 AND y>=0 THEN  GO T
O 230
 300 REM finished
 310 PAPER 7: INK 0: BORDER 7
1000 REM PICTURE CO-ORDINATES
1010 DATA 40,70,90,70,90,80,110,
80,110,90
1020 DATA 140,90,145,120,160,120
,155,90,165,90
1030 DATA 165,70,200,70,195,40,6
0,40,40,70
1040 DATA -1,-1
```

Program 2.6

Many drawings consist of several groups of points. Each group will contain a number of points which should be joined in order, but individual groups are not joined to one another.

Refining the technique a little further allows drawings like this to be produced just as easily. This time, reading *x* and *y* co-ordinates of 1000,1000 signals the end of a group, so the next command must be a PLOT, to position the first point in the next group.

Example 2.7

```
 100 REM FELICITATIONS
 110 REM set colours
 120 PAPER 7: INK 4: BORDER 7
 200 REM join points
 210 READ x,y
 220 PLOT x,y
 230 READ xn,yn
 240 IF xn<0 OR yn<0 THEN  GO TO
 300
 250 IF xn=1000 OR yn=1000 THEN
 GO TO 210
 260 DRAW xn-x,yn-y
 270 LET x=xn
 280 LET y=yn
 290 IF x>=0 AND y>=0 THEN  GO T
```

Program 2.7 *(continues)*

```
0 230
 300 REM finished
 310 PAPER 7: INK 0: BORDER 7
1000 REM PICTURE CO-ORDINATES
1010 DATA 112,35,80,50,70,65,70,
75,80,90
1020 DATA 90,120,70,150,110,130,
149,130,189,150
1030 DATA 169,120,174,90,184,75,
184,65,174,50
1040 DATA 142,35,112,35,1000,100
0
1050 DATA 90,70,164,70,149,55,10
5,55,90,70,1000,1000
1060 DATA 97,63,157,63,1000,1000
1070 DATA 105,55,105,70,1000,100
0
1080 DATA 116,55,116,70,1000,100
0
1090 DATA 127,55,127,70,1000,100
0
1100 DATA 138,55,138,70,1000,100
0
1110 DATA 149,55,149,70,1000,100
0
1120 DATA 112,85,117,80,127,80,1
27,107,127,80
1130 DATA 137,80,142,85,1000,100
0
1140 DATA 60,50,80,65,1000,1000
1150 DATA 50,70,80,70,1000,1000
1160 DATA 60,85,80,75,1000,1000
1170 DATA 174,65,194,50,1000,100
0
1180 DATA 174,70,204,70,1000,100
0
1190 DATA 174,75,194,85,1000,100
0
1200 DATA 103,108,109,112,115,10
8,109,104,103,108,1000,1000
1210 DATA 139,108,145,112,151,10
8,145,104,139,108,1000,1000
1220 DATA 108,108,109,109,110,10
8,109,107,108,108,1000,1000
1230 DATA 144,108,145,109,146,10
8,145,107,144,108,-1,-1
```

Program 2.7

CIRCLES

The CIRCLE command provides a quick, neat way of drawing circles. Sometimes, the circle need not be drawn, but it is necessary to identify the position of various points on the circumference so

that these can be used to draw another figure. Many regular shapes can be drawn by joining points which lie on the circumference of a circle. A square, for instance, can be drawn as illustrated in Fig. 2.2, by joining four points on the circumference of a suitably sized circle which are separated by equal intervals (a quarter of the circumference, in this case).

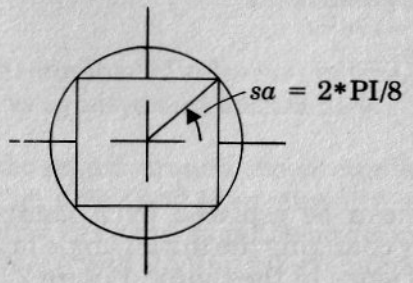


Figure 2.2 A square drawn inside a circle

One complete revolution is 2*PI radians, so each of the angles marked in the diagram is 2*PI/8 radians. If the length of each side of the square is *sl*, the diagonal of the square is the radius of the circle, where

$$r = \text{SQR}\ [(sl/2)\uparrow 2 + (sl/2)\uparrow 2]$$

If the (anticlockwise) angle between the horizontal and the first diagonal of the square is *sa*, where *sa* = 2*PI/8, then the co-ordinates of the first corner of the square are

$$x = r*\text{COS}\ sa$$
$$y = r*\text{SIN}\ sa$$

The angle increases by 2*PI/4 between each diagonal.

The following example shows how to plot a square using this technique.

Example 2.8

```
100 REM PLOT SQUARE IN CIRCLE
110 REM set colours
120 PAPER 7: INK 0: BORDER 7
130 CLS
200 REM initialise variables
210 LET sx=126
220 LET sy=88
230 LET sl=50
240 LET sa=2*PI/8
300 REM draw square
```

Program 2.8 *(continues)*

```
 310 LET r=SQR ((sl/2)^2+(sl/2)^
2)
 320 LET x=sx+r*COS sa
 330 LET y=sy+r*SIN sa
 340 PLOT x,y
 350 FOR i=3 TO 9 STEP 2
 360 LET sa=(2*PI/8)*i
 370 LET xn=sx+r*COS sa
 380 LET yn=sy+r*SIN sa
 390 DRAW xn-x,yn-y
 400 LET x=xn
 410 LET y=yn
 420 NEXT i
```

Program 2.8

The square can be replaced by a many-sided figure called a POLYGON, by changing the initial angle to 2*PI/(2**ns*) where *ns* is the number of sides in the figure. Figure 2.3 shows that the angle between the diagonals becomes 2*PI/*ns*.

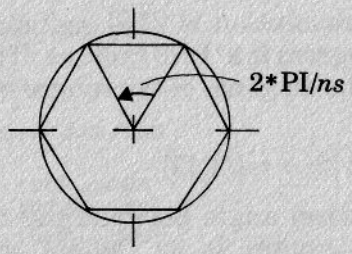


Figure 2.3 A polygon drawn inside a circle

Example 2.9

```
 100 REM PLOT POLYGON
 110 REM set colours
 120 PAPER 7: INK 0: BORDER 7
 130 CLS
 200 REM initialise variables
 210 LET sx=126
 220 LET sy=88
 230 LET sl=50
 240 LET ns=8
 250 LET sa=2*PI/(2*ns)
 300 REM draw square
 310 LET r=SQR ((sl/2)^2+(sl/2)^
2)
 320 LET x=sx+r*COS sa
 330 LET y=sy+r*SIN sa
 340 PLOT x,y
 350 FOR i=3 TO 2*ns+1 STEP 2
```



```
 360 LET sa=(2*PI/(2*ns))*i
 370 LET xn=sx+r*COS sa
 380 LET yn=sy+r*SIN sa
 390 DRAW xn-x,yn-y
 400 LET x=xn
 410 LET y=yn
 420 NEXT i
```

Program 2.9

Experiment with the number of sides (*ns*) in line 240. A common figure appears when *ns* is increased sufficiently.

Instead of joining the points around the circumference in order, they could be held in an array and then joined in a different way. The next figure is formed by calculating the position of a number of points on a circumference and joining every point to the point 8 positions further round the circumference.

Example 2.10

```
100 REM JOIN POINTS ON CIRCLE
110 REM set colours
120 PAPER 7: INK 0: BORDER 7
130 CLS
200 REM initialise variables
210 LET sx=126
220 LET sy=88
230 LET r=85
240 LET ns=24
250 DIM x(ns)
260 DIM y(ns)
300 REM create points
310 FOR i=1 TO ns
320 LET a=(2*PI/ns)*i
330 LET x(i)=sx+r*COS a
340 LET y(i)=sy+r*SIN a
350 NEXT i
400 REM join points
410 FOR i=1 TO ns
420 LET p=i+8
430 IF p>ns THEN  LET p=p-ns
440 PLOT x(i),y(i)
450 DRAW x(p)-x(i),y(p)-y(i)
460 NEXT i
```

Program 2.10

Experiment with different values of *ns* in line 240. Values less than 8 will not work, because of the 8 in line 420. Exactly 8 produces an unexpected result, while values of 50 and over produce a rather different looking figure.

THE ELLIPSE

An ellipse looks rather like a 'squashed' circle. A typical example is shown in Fig. 2.4.

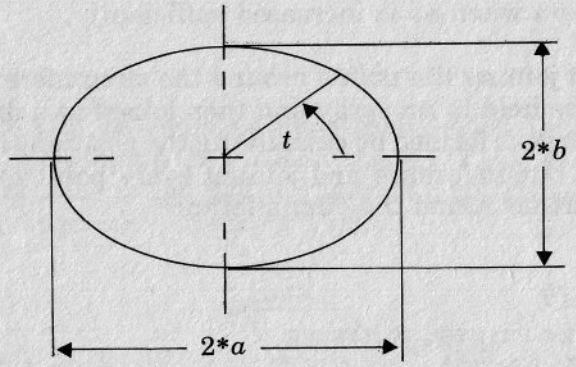


Figure 2.4 A typical ellipse

The equation of an ellipse depends on the lengths of the major and minor axes. If the length of the major axis is 2**a*, and the length of the minor axis is 2**b*, then the *x* and *y* co-ordinates of any point on the ellipse can be found using

$x = a*\text{COS}\ t$
$y = b*\text{SIN}\ t$

where *t* is an angle (in radians) measured anticlockwise from the horizontal.

Example 2.11

```
100 REM ELLIPSE
110 REM set colours
120 PAPER 7: INK 0: BORDER 7
130 CLS
200 REM initialise variables
210 LET cx=126
220 LET cy=88
230 LET a=60: REM major axis
240 LET b=30: REM minor axis
300 REM draw ellipse
310 LET x1=a: LET y1=0: REM ini
tial point
320 PLOT cx+x1,cy+y1
330 FOR t=0 TO 2*PI+.1 STEP 0.1
340 LET x2=INT (a*COS t+.5)
350 LET y2=INT (b*SIN t+.5)
360 DRAW x2-x1,y2-y1
370 LET x1=x2
380 LET y1=y2
390 NEXT t
```

Program 2.11

Different ellipses are produced using different values of *a* and *b* in lines 230 and 240.

Lines 340 and 350 calculate *x* and *y* co-ordinates, but also round these for the benefit of the DRAW command in line 360. Without this rounding, there may be a misalignment when the last point is joined to the first point on the ellipse.

Line 330 takes the angle *t* a little beyond the 2*PI, to ensure that joining takes place between the last and first points.

The ellipse may seem like a rather mathematical shape which has little application to computer graphics. This is not so, as the next example will show. The ellipse possesses at least one valuable property – it represents a circle lying at an angle to the viewer. Think of a coin with a circular shape; then think of the outline of the coin if it is tilted backwards, between the face-on and the edge-on positions. Viewing the coin from a distance will show that its outline is now an ellipse. This effect allows the simulation of a three-dimensional effect since it can be used to add a third dimension (depth) to a circle drawn on the screen.

Example 2.12

The Program 2.12 is a general-purpose program which can be used to draw three-dimensional representations of objects on the screen.

```
100 REM SKELETON SOLID
110 REM set colours
120 PAPER 7: INK 0: BORDER 7
130 CLS
200 REM initialise variables
210 LET inc=24: REM no. of sect
ions
220 LET f=0.25: REM ratio of a
to b (axes of ellipse)
```

Program 2.12 *(continues)*

```
 300 REM store points in arrays
 310 READ n: REM no. of points
 320 DIM x(n): DIM y(n)
 330 FOR i=1 TO n
 340 READ x(i),y(i)
 350 NEXT i
 400 REM draw ellipses
 410 FOR i=2 TO n-1
 420 LET x1=x(i)-x(n): LET y1=0:
 REM initial point
 430 PLOT x(i),y(i)
 440 FOR t=0 TO 2*PI+.1 STEP .1
 450 LET x2=INT ((x(i)-x(1))*COS
 t+.5)
 460 LET y2=INT ((x(i)-x(1))*f*S
IN t+.5)
 470 DRAW x2-x1,y2-y1
 480 LET x1=x2
 490 LET y1=y2
 500 NEXT t
 510 NEXT i
 600 REM draw axis
 610 PLOT x(1),y(1)
 620 DRAW x(n)-x(1),y(n)-y(1)
 700 REM draw sections
 710 FOR t=0 TO 2*PI STEP 2*PI/i
nc
 720 PLOT x(1),y(1)
 730 FOR i=2 TO n
 740 DRAW (x(i)-x(1))*COS t-(x(i
-1)-x(1))*COS t,y(i)+(x(i)-x(1))
*f*SIN t-(y(i-1)+(x(i-1)-x(1))*f
*SIN t)
 750 NEXT i
 760 NEXT t
1000 REM CO-ORDINATES OF CROSS S
ECTION
1010 DATA 7: REM no. of points
1020 DATA 126,125
1030 DATA 195,125
1040 DATA 235,85
1050 DATA 235,55
1060 DATA 170,55
1070 DATA 170,15
1080 DATA 126,15
```

Program 2.12

This program can easily be modified to draw other objects. Begin by defining the right-hand half of a cross-section as a series of points which are to be joined in order. Figure 2.5 shows an example.

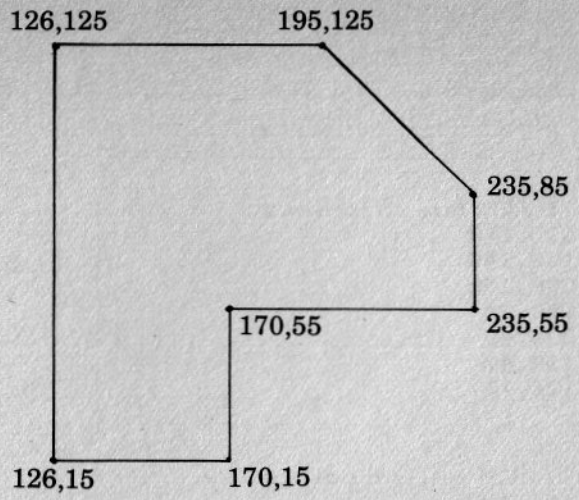


Figure 2.5 A right-hand half cross-section

These points should be entered in the program as DATA in lines 1020 onwards. The first point should be the top point on the central axis, and the last point should be the bottom point on the central axis. The points should be entered in order, and the total number of points should be entered as DATA in line 1010.

The program draws an ellipse using each point except the first and last. Each ellipse is drawn so that a is the distance from the central axis to the point $x(i) - x(1)$, or $x(i) - x(n)$. Instead of using values of a and b for each ellipse, a and b are related by the factor f (representing the eccentricity of the ellipse and hence the angle of tilt of the circle which the ellipse represents). Lines 400–510 draw the ellipses.

The central axis is then drawn using lines 610 and 620.

The sections through the object are drawn by considering each point on the given section, and changing its co-ordinates as the angle of turn, t, is swept through 0 to 2*PI radians.

Each point is considered to be moving round an ellipse (the ellipses drawn at the start of the program). The ellipses then represent the corners or edges where there is a change of direction between faces of the object.

The illusion can be changed in two main ways:

1. By changing the angle of tilt (change the value of f in line 220).
2. By changing the number of sections drawn (change the value of *inc* in line 210).

Programs 2.C, 2.D and 2.E show some other sets of DATA to try with the SKELETON SOLID program.

```
1010 DATA 4: REM no. of points
1020 DATA 126,150
1030 DATA 200,150
1040 DATA 200,30
1050 DATA 126,30
```

Program 2.C

```
1010 DATA 7: REM no. of points
1020 DATA 126,170
1030 DATA 150,140
1040 DATA 200,140
1050 DATA 250,90
1060 DATA 200,40
1070 DATA 150,40
1080 DATA 126,70
```

Program 2.D

```
1010 DATA 8: REM no. of points
1020 DATA 126,140
1030 DATA 200,140
1040 DATA 200,120
1050 DATA 170,120
1060 DATA 170,50
1070 DATA 200,50
1080 DATA 200,30
1090 DATA 126,30
```

Program 2.E

CURVES FROM LINES

Curved patterns can be produced from a series of straight lines. The basic technique is rather like the method used to make curved stitching pictures.

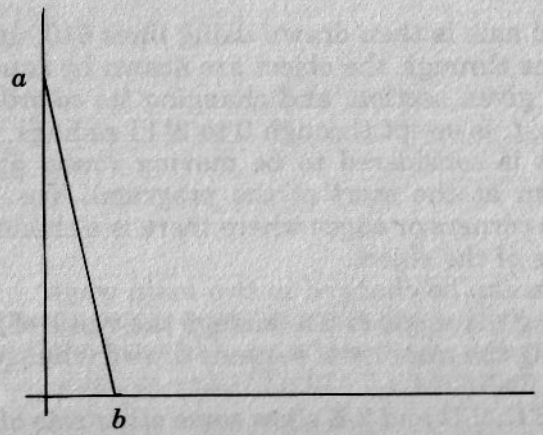


Figure 2.6 Drawing an initial line

Start by drawing a line between two points, *a* and *b*, as shown in Fig. 2.6. Then lower the point *a*, and move point *b* to the right, to achieve the effect shown in Fig. 2.7.

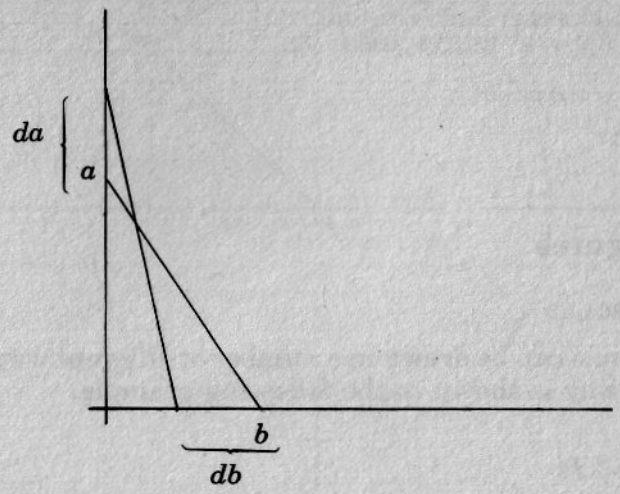


Figure 2.7 Drawing a second line

Repeat the process many times.

Point *a* is lowered the same distance each time, and point *b* moves the same distance to the right each time (although the distances *a* and *b* move need not be the same). This is easily programmed, using *da* and *db* to represent the distances *a* and *b* move respectively.

Example 2.13

```
100 REM CURVES FROM LINES
110 REM set colours
120 PAPER 7: INK 0: BORDER 7
130 CLS
200 REM draw axes
210 PLOT 0,0: DRAW 0,175
220 PLOT 0,0: DRAW 255,0
300 REM initialise variables
310 REM set distance to a
320 LET a=175
330 REM set reduction in a
340 LET da=5
350 REM set distance to b
360 LET b=5
```

Program 2.13 *(continues)*

```
 370 REM set increase in b
 380 LET db=7
 400 REM draw lines
 410 PLOT 0,a
 420 DRAW b,-a
 430 LET a=a-da: LET b=b+db
 440 IF a>0 AND b<255 THEN  GO T
O 410
 500 REM finished
```

Program 2.13

Solid figures

SOLID CIRCLES

A solid circle can be drawn in a number of different ways. The most effective way is shown in the following example.

Example 2.14

```
  10 REM SOLID CIRCLE
  20 CLS
 100 LET r=50
 110 LET cx=125
 120 LET cy=87
 200 FOR h=0 TO r
 210 LET xc=INT (SQR (r^2-h^2)+.
5)
 220 PLOT cx-xc,cy+h
 230 DRAW 2*xc,0
 240 PLOT cx-xc,cy-h
 250 DRAW 2*xc,0
 260 NEXT h
```

Program 2.14

Values of h from 0 to r are used, where h represents the vertical height above the centre of the circle. Figure 2.8 shows the important points on the circle, for a typical value of h.

Here h and r are used to calculate the distance xc, in line 210. The beginning of the current horizontal line is fixed at $cx - xc, cy + h$ in line 220. The shading in the upper part of the circle is drawn in line 230.

Lines 240 and 250 draw the shading in the lower part of the circle.

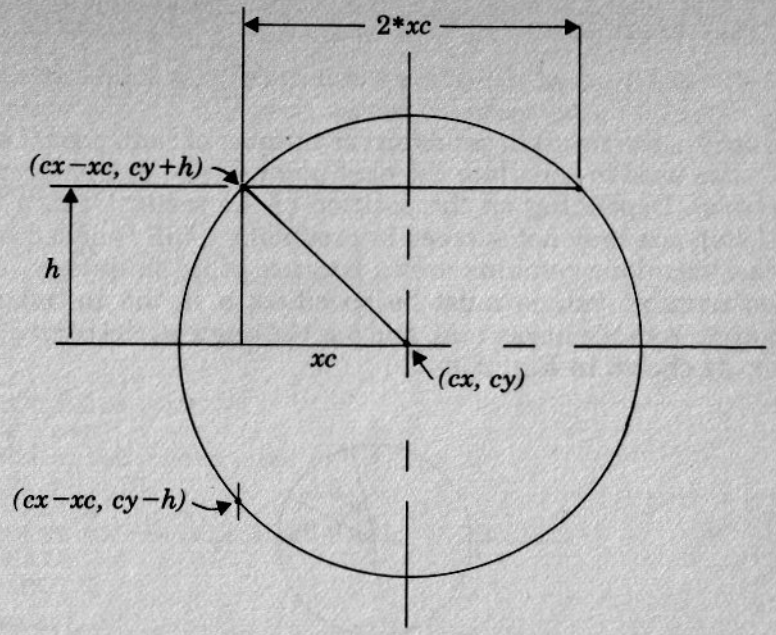


Figure 2.8 Important points on the circle

Another technique involves drawing a large number of radii inside a circle. The end result depends on the number of radii used.

Example 2.15

The different patterns produced by drawing different numbers of radii inside a circle are shown in Program 2.15.

```
  10 REM REPEATED PATTERN
  20 CLS
  30 LET cx=126: LET cy=88
  40 LET r=87
  50 FOR s=1 TO 5
  60 GO SUB 100
  70 PAUSE 100: CLS
  80 NEXT s
  90 STOP
 100 FOR a=0 TO 6.28 STEP 6.28/(
s*r)
 110 PLOT cx,cy: DRAW r*COS a,r*
SIN a
 120 NEXT a
 130 RETURN
```

Program 2.15

Restricting the size of the angle when drawing a solid circle using many radii produces a shaded sector.

As before, the result depends on the number of radii used (i.e., the STEP size used to calculate the next point on the circumference of the circle). Depending on the position of the sector, even a fairly small step size may not succeed in producing a fully shaded sector, but the technique remains useful because of its simplicity.

Two angular values must be specified; *a* is the initial angle measured from the horizontal, and *b* is the angle at the centre of the sector, as shown in Fig. 2.9.

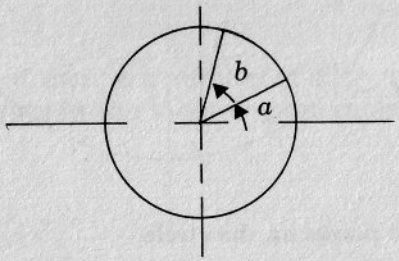


Figure 2.9 Angles *a* and *b*

Example 2.16

```
100 REM SOLID SECTOR
110 REM set colours
120 PAPER 7: INK 3: BORDER 7
130 CLS
200 REM initialise variables
210 LET r=50
220 LET a=0.25*PI
230 LET b=0.125*PI
240 LET cx=126
250 LET cy=88
260 CIRCLE cx,cy,r
300 REM draw sector
310 FOR t=a TO a+b STEP 2*PI/(3
6*r)
320 PLOT cx,cy
330 DRAW r*COS t,r*SIN t
340 NEXT t
400 REM finished
410 PAPER 7: INK 0: BORDER 7
```

Program 2.16

Rectangles can sometimes be produced by printing SPACE characters using a different colour of paper.

Example 2.17

```
100 REM PAPER RECTANGLE
110 REM set colours
120 PAPER 7: INK 0: BORDER 7
130 CLS
200 REM define variables
210 LET r$="        "
220 LET row=10
230 LET column=12
240 LET b=5
300 REM print rectangle
310 FOR h=0 TO b
320 PAPER 2
330 PRINT AT row+h,column;r$
340 NEXT h
350 PAPER 7
```

Program 2.17

The upper left corner of the rectangle is fixed at row, column and the loop beginning at line 310 steps through *h* rows printing a pre-defined character consisting of 8 spaces which will appear as solid colour, since line 320 ensures that the colour of the paper is different from the colour originally used when the screen was cleared.

Example 2.18

This example uses the technique of example 2.17 to produce rectangles, but assigns some of the parameters randomly before printing the rectangles. The effect is rather pleasing

```
100 REM PAPER RECTANGLES
110 REM set colours
120 PAPER 7: INK 0: BORDER 7
130 CLS
200 REM define character
210 LET r$="        "
250 REM produce rectangles
260 FOR i=1 TO 100
300 REM generate parameters
310 LET b=INT (10*RND)
320 LET row=INT (11*RND)
330 LET column=INT ((32-LEN r$-
b)*RND)
```

Program 2.18 *(continues)*

```
340 PAPER INT (7*RND)
400 REM print rectangle
410 FOR h=0 TO b
420 PRINT AT row+h,column;r$
430 NEXT h
440 NEXT i
500 REM finished
510 PAPER 7: INK 0: BORDER 7
```

Program 2.18

Using INK, solid rectangles can be constructed by drawing groups of lines. Figure 2.10 shows the important values used when constructing such a rectangle.

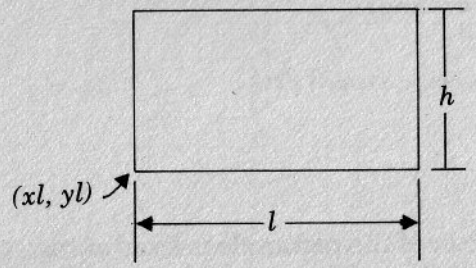


Figure 2.10 Starting point and height of a solid rectangle

Example 2.19

```
100 REM DRAW SOLID RECTANGLE
110 REM set colours
120 PAPER 7: INK 0: BORDER 7
130 CLS
200 REM initialise variables
210 LET l=30: REM length
220 LET h=15: REM height
230 LET x1=110: REM bottom left
240 LET y1=80: REM bottom left
300 REM draw rectangle
310 FOR y=y1 TO y1+h
320 PLOT x1,y
330 DRAW l,0
340 NEXT y
```

Program 2.19

3 DRAGON'S MOUTH

Once upon a time, in the dim and distant past, when men wore hairy vests and Spectrums hadn't been invented, there lived a wise little old man named MIKA.

Dragons ruled the land in those days, and MIKA was employed in a dragon's kitchen where he pickled goblins before feeding them to the dragon. The dragon lived in an underground labyrinth of many rooms, and every day MIKA had to take a pickled goblin to the dragon, for lunch. Unfortunately, goblins are leaky little fellows, and they drip blood everywhere. Goblins' blood is magic, and it dissolves wise little old men like MIKA, so he must avoid stepping on any of it before he has fed the dragon. The blood is easy to see – it is bright red.

At lunchtime, the dragon could be anywhere, but wherever he is MIKA must come from the kitchen and drag the goblin through every room in the labyrinth before he jumps into the dragon's mouth to deliver lunch.

Use keys 5, 6, 7 and 8 to move.

```
  10 REM DRAGON'S MOUTH
  20 REM SET COLOURS
  30 PAPER 7: INK 0: BORDER 7
  40 OVER 0: CLS
 100 REM LOAD MACHINE CODE ROUTI
NES
 110 CLEAR 32494: REM OK FOR 16K
 OR 48K
 120 PRINT  INK 2;AT 11,8;"DRAGO
N'S MOUTH"
 130 LET start1=32495: REM SET A
TTRIBUTES AT PRINT POSITION
 140 LET start2=start1+49: REM D
ECODE ATTRIBUTES
 150 LET s=start1
 160 LET length=56+49
 170 FOR i=1 TO length
 180 READ n
 190 POKE s,n
 200 LET s=s+1
 210 NEXT i
 400 REM SET LEVEL
 410 LET level=0
 420 LET level=level+1
 450 IF level=4 THEN  GO SUB 640
0: GO TO 7000
```

Program 3.1 *(continues)*

```
 460 RESTORE 5000
 470 FOR i=1 TO level
 480 READ n,p
 490 NEXT i
 500 REM
 510 LET cl=11
 520 LET cc=15
 530 IF n/2<>INT (n/2) THEN  LET
 pl=cl+1+3*INT (n/2): LET pc=cc+
1+3*INT (n/2)
 540 IF n/2=INT (n/2) THEN  LET
pl=cl-1+3*INT (n/2): LET pc=cc-1
+3*INT (n/2)
 550 LET cx=8*(pc+1)-1
 560 LET cy=176-8*(pl+1)
 600 REM DRAW ROOMS
 605 CLS
 610 PLOT cx,cy
 620 DRAW -24*n,0: DRAW 0,24*n:
DRAW 24*n,0: DRAW 0,-24*n
 630 FOR i=1 TO n-1
 640 PLOT cx-24*i,cy
 650 DRAW 0,24*n
 660 NEXT i
 670 FOR i=1 TO n-1
 680 PLOT cx,cy+24*i
 690 DRAW -24*n,0
 700 NEXT i
 710 PLOT cx,cy
 720 DRAW 0,-24: DRAW -24,0: DRA
W 0,24
 750 REM DOORS
 760 INK 7
 770 FOR i=1 TO n
 780 FOR j=1 TO n-1
 790 PRINT AT pl-1-3*(i-1),pc-3*
j;CHR$ 143
 800 NEXT j
 810 NEXT i
 820 FOR i=1 TO n-1
 830 FOR j=1 TO n
 840 PRINT AT pl-3*i,pc+2-3*j;CH
R$ 143
 850 NEXT j
 860 NEXT i
 870 PRINT AT pl,pc-1;CHR$ 143
 880 PRINT AT pl-3*n,pc-1-3*p;CH
R$ 143
 890 PAPER 7: INK 0
 900 REM SET CURRENT CO-ORDS
 910 LET l=pl+2
 920 LET c=pc-1
 930 REM SET EXIT CO-ORDS
 940 LET fl=pl-3*n
 950 LET fc=pc-1-3*p
```

Program 3.1 *(continues)*


```
 960 REM DRAW DRAGON
 970 LET dx=cx-12-24*p: LET dy=c
y+24*n+10
 980 PLOT dx,dy: DRAW -3,-2: DRA
W -3,1: DRAW -5,0: DRAW 5,8: DRA
W -7,0: DRAW 9,5: DRAW 10,0: DRA
W 9,-5: DRAW -7,0: DRAW 5,-8: DR
AW -5,0: DRAW -3,-1: DRAW -3,2:
PLOT dx-4,dy-2: DRAW 0,9: DRAW 5
,1: DRAW 5,-1: DRAW 0,-9: DRAW -
9,0
 990 PLOT dx-4,dy+9: DRAW 0,2: D
RAW 10,0: DRAW 0,-2: DRAW -10,0:
 PLOT dx+1,dy+9: DRAW 0,2
1000 REM MAIN LOOP
1010 OVER 1
1020 REM PREPARE TO DECODE ATTRI
BUTES
1022 POKE 23296,l
1024 POKE 23297,c
1026 RANDOMIZE USR start2: REM D
ECODE ATTRIBUTES
1028 IF PEEK 23300=2 THEN  GO TO
 3500: REM STEPPED INTO A RED RO
OM
1029 PRINT AT l,c;CHR$ 42
1030 LET dl=0: LET dc=0
1040 LET k$=INKEY$
1050 IF k$="" THEN  GO TO 1040
1060 IF k$="5" THEN  LET dc=-3
1070 IF k$="8" THEN  LET dc=3
1080 IF k$="7" THEN  LET dl=-3
1090 IF k$="6" THEN  LET dl=3
1100 LET l=l+dl
1110 LET c=c+dc
1120 IF (l=pl-1-3*n) AND (c=pc-1
-3*p) THEN  GO TO 3000: REM SUCC
ESS
1130 IF (c=pc-1) AND (l=pl+2) TH
EN  GO TO 1190: REM STILL AT STA
RT
1140 IF (c>pc) OR (c<pc-1-3*(n-1
)) OR (l>pl) OR (l<pl-1-3*(n-1))
 THEN  LET l=l-dl: LET c=c-dc: G
O TO 1030: REM OUT OF BOUNDS
1190 PRINT AT l-dl,c-dc;CHR$ 42
1200 GO SUB 2000: REM CHANGE PAP
ER TO RED
1210 GO TO 1020
2000 REM GIVE ROOM RED PAPER AND
 INK
2002 POKE 23298,0: REM FLASH
2003 POKE 23299,0: REM BRIGHT
2004 POKE 23300,2: REM PAPER
2005 POKE 23301,2: REM INK
```

```
2010 FOR i=-1 TO 1
2020 FOR j=-1 TO 1
2030 POKE 23296,l-dl+i
2040 POKE 23297,c-dc+j
2060 RANDOMIZE USR start1: REM S
ET PAPER COLOUR
2070 NEXT j
2080 NEXT i
2090 RETURN
3000 REM SUCCESS
3010 PRINT AT l-dl,c-dc;CHR$ 42
3015 GO SUB 2000
3020 PRINT AT l,c;CHR$ 42
3025 OVER 0
3030 GO SUB 6300: REM CHECK ALL
ROOMS ARE RED
3040 IF r<>0 THEN  GO TO 3100
3050 GO SUB 6000: REM WELL DONE
3060 GO SUB 6200: REM FETCH KEYP
RESS
3070 IF k$="y" THEN  GO TO 420:
REM NEXT LEVEL
3080 IF k$<>"y" THEN  GO TO 7000
3100 GO SUB 6100: REM MISSED ROO
MS
3110 GO SUB 6200: REM FETCH KEYP
RESS
3120 IF k$="y" THEN  GO TO 460:
REM SAME LEVEL
3130 IF k$<>"y" THEN  GO TO 7000
3500 REM STEPPED INTO A RED ROOM
3502 OVER 0
3510 PRINT AT 20,0;"SQUELCH! YOU
 STEPPED INTO A RED ROOM AND DIS
SOLVED! - GAME OVER!"
3520 PAPER 7: INK 0: BORDER 7: G
O TO 8000
3530 GO TO 7000
4000 REM SET ATTRIBUTES AT PRINT
 POSITION
4010 DATA 33,0,91,126,35,94,254,
22,48,34
4020 DATA 33,0,88,22,0,25,17,32,
0,254
4030 DATA 0,40,4,25,61,24,248,23
5,33,2
4040 DATA 91,62,0,174,7,35,174,7
,7,7
4050 DATA 35,174,7,7,7,35,174,18
,201
4100 REM DECODE ATTRIBUTES AT PR
INT POSITION
4110 DATA 33,0,91,126,35,94,254,
22,48,54
4120 DATA 33,0,88,22,0,25,17,32,
```

Program 3.1 *(continues)*


```
0,254
4130 DATA 0,40,4,25,61,24,248,17
,2,91
4140 DATA 126,7,230,1,18,19,126,
7,7,230
4150 DATA 1,18,19,126,15,15,15,2
30,7,18
4160 DATA 19,126,230,7,18,201
5000 REM ROOMS AND EXIT
5010 DATA 3,0
5020 DATA 4,1
5030 DATA 4,3
6000 REM WELL DONE
6010 PRINT AT 20,0;"WELL DONE! Y
OU FED THE DRAGON!"
6020 PRINT AT 21,0;"TRY THE NEXT
 LEVEL? (y or n)"
6030 RETURN
6100 REM MISSED ROOMS
6110 PRINT AT 20,0;"OOPS! YOU MI
SSED ";r;" ROOMS."
6120 PRINT AT 21,0;"TRY AGAIN? (
y or n)"
6130 RETURN
6200 REM GET KEYPRESS
6210 LET k$=INKEY$
6220 IF k$="" THEN  GO TO 6210
6230 RETURN
6300 REM CHECK ALL ROOMS ARE RED
6310 LET r=0: REM NUMBER OF ROOM
S NOT RED
6320 FOR i=0 TO n-1
6330 FOR j=0 TO n-1
6332 POKE 23296,p1-1-3*j
6334 POKE 23297,pc-1-3*i
6336 RANDOMIZE USR start2
6340 IF PEEK 23300<>2 THEN  LET
r=r+1
6350 NEXT j
6360 NEXT i
6370 RETURN
6400 REM COMPLETED ALL LEVELS
6410 PAPER 7: INK 0: BORDER 7
6420 CLS
6430 PRINT AT 2,0;"YOU HAVE SUCC
ESSFULLY COMPLETED"
6440 PRINT AT 4,0;"ALL THE LEVEL
S."
6450 PRINT AT 6,0;"YOU ARE HEREB
Y INVITED TO JOIN"
6460 PRINT AT 8,0;"THE RANKS OF
THE EXCLUSIVE"
6470 PRINT  INVERSE 1;AT 10,4;"G
RAND DRAGON MASTER CLUB"
6480 PRINT AT 12,0;"YOU COULD AL
```

```
WAYS TRY ADDING"
6490 PRINT AT 14,0;"SOME OTHER L
EVELS."
6500 PAUSE 100
7000 REM END
7010 PAPER 7: INK 0: BORDER 7
7020 OVER 0: CLS
7030 PRINT AT 11,13;"BYE!"
```

Program 3.1

So much for the fairy tales; how does the game really work? The object is to move from the start to the dragon's mouth, passing through each room exactly once.

The * symbol represents the wise little old man, and when it is moved out of a room the INK and PAPER inside that room are turned red. Before placing the * inside a room, the attributes are examined. If the PAPER in the room is red, the game ends.

Two machine code routines are used:

(a) Set attributes at a PRINT position.
This is used to give the print positions inside a room the attributes
FLASH 0: BRIGHT 0: PAPER 2: INK 2
which will turn the room red.

(b) Decode attributes at a PRINT position.
This is used to determine whether or not a room is red.

Program description

10– 40 — Set PAPER to white and INK to black.

100–210 — Load routines.
SET ATTRIBUTES AT PRINT POSITION begins at start1. DECODE ATTRIBUTES AT PRINT POSITION begins at start2. Setting RAMTOP to 32494 allows the program to be used with 16K or 48K machines.

400–520 — Set the skill level. The maximum level is determined by the data held in lines 5000 onwards. Each level requires two items of data.
(a) The size of the grid representing the rooms.
In line 5010 this is 3, so the grid is 3 × 3, i.e., it holds 9 rooms. The projecting room at the bottom right, in which the * is placed at the start is added after the grid is drawn and does not count as one of the rooms defined in the data.
(b) The position of the exit. The rooms on the top row are numbered from 0 (at the rightmost end), as shown in Fig. 3.1.

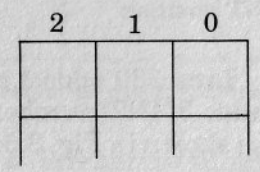


Figure 3.1 Numbering of the rooms on the top row

The loop from 470 to 490 ensures that 'level' pairs of data items are read and that the 'levelth' pair are assigned to *n* and *p* on exit.

510–520 — *cl* and *cc* represent the line and column of the PRINT position at the centre of the grid of rooms.

530–540 — *pl* and *pc* represent the line and column of the bottom right PRINT position inside the grid (see Fig. 3.2).

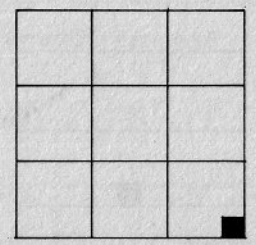


Figure 3.2 The bottom right PRINT position

Every room contains 3 × 3 PRINT positions, regardless of the number of rooms.

When the grid contains an odd number of rooms, Fig. 3.3 shows that *cl,cc* represents the PRINT position in the centre of the centre room.

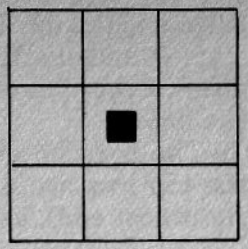


Figure 3.3 The centre PRINT position

Line 530 adds 1 to *cl*, (see Fig. 3.4) then adds 3*INT(number of rooms/2) to find *pl*, as shown in Fig. 3.5; *pc* is found in a similar way.

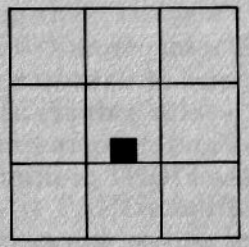


Figure 3.4 Bottom centre PRINT position in the centre room

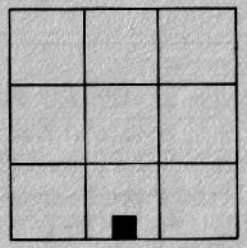


Figure 3.5 Bottom centre PRINT position in the bottom centre room

When the grid contains an even × even number of rooms *cl,cc* represents the PRINT position whose top left corner is at the central intersection of the walls of the rooms, as shown in Fig. 3.6.

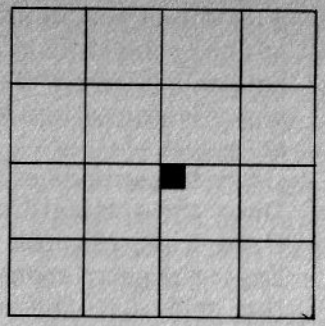


Figure 3.6 Starting position in an even × even grid

pl is found by subtracting 1 from *cl* (see Fig. 3.7), and adding 3*INT(number of rooms/2) (see Fig. 3.8).

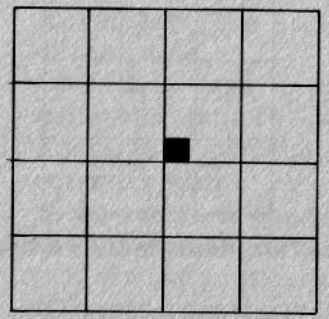


Figure 3.7 The effect of subtracting 1 from *cl*

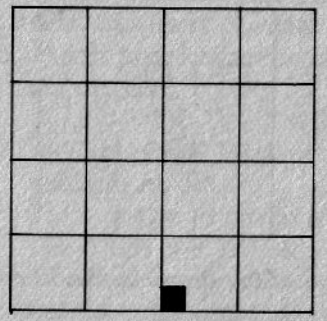


Figure 3.8 The position indicated by the final value of *pl*

pc is found in a similar way.

550–560 *cx* and *cy* are the *x* and *y* co-ordinates of the bottom left corner of the PRINT position at *pl,pc*. So, the bottom left corner of the grid is at *cx,cy*.

600–620 Draw the outside of the grid.

630–660 Draw the vertical lines inside the grid.

670–700 Draw the horizontal lines inside the grid.

710–720 Draw the extra room outside the bottom of the grid, i.e., the room in which the * initially appears.

750–890 The doors are produced by printing CHR$ 143 (a solid graphics character).

760–810 produce the doors in the vertical room dividers, as shown in Fig. 3.9.

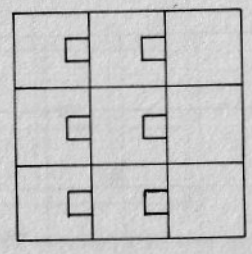


Figure 3.9 The position of the doors in the vertical room dividers

820–860 produce the doors in the horizontal room dividers, as shown in Fig. 3.10.

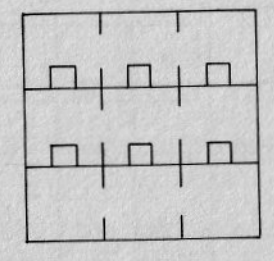


Figure 3.10 The position of the doors in the horizontal room dividers

870 produces the door leading into the starting room.

880 produces the exit.

890 resets the INK and PAPER colours to white PAPER and black INK.

900–920 The current line and column (*l,c*) represent the present position of the * symbol. These are set to represent the centre of the room in which the * first appears. The * is always placed in the centre of a room, so movement is from the centre of one room to the centre of the next room.

930–950 The line and column co-ordinates of the dragon's mouth are set to *fl,fc*.

960–990 The dragon's head is drawn. *dx* and *dy* represent the bottom of the dragon's chin. Plotting and drawing takes place relative to *dx,dy* to allow the head to be placed opposite the exit door.

1000–1210 The main loop controls the keyboard input, movement of the *, and analysis of the state of the game.

1010 OVER1 is used to print and erase the * using the same PRINT command.

1020–1026 Examine the attributes of the centre of the current room.

1028 If the centre of the room has red PAPER, the * is about to be placed in a red room, so exit via the routine at 3500.

1029 If the centre of the room is not red, print the *.

1030 *dl* and *dc* represent the change to be made to the current line and column values, to find the next position for the *.

1040–1110 Fetch any keypress and change *l* and *c* as required.

1120 If *l,c* = *fl,fc* the * will reach the dragon's mouth when next printed, so exit via the routine at 3000.

1130 If *l,c* is the original starting position (i.e., the * has not moved) continue at 1190.

1140 Check whether the latest move places *l,c* outside the grid. If so, do not allow the move. Restore *l* and *c* to their former values. Line 1130 is necessary because it simplifies the check for out of bounds in line 1140.

1190 Remove the * from its present position by

printing it with OVER1 (set in line 1010).

1200 Change the room just left, to red, by setting PAPER and INK to red.

1210 Repeat from 1020.

2000–2090 This subroutine uses a pair of nested loops, with the set attributes routine, to turn PAPER and INK red inside a room.

3000–3130 If the * will reach the dragon's mouth, this section deals with possible success.

3010–3025 The last room is turned red; the * is removed and placed in the dragon's mouth. OVER0 is set, to enable normal printing to resume.

3030 Success will be achieved if all the rooms in the grid are red, implying that they have all been visited once. The subroutine at 6300 is used to return a value of *r*, representing the number of rooms which are not red.

3040 If there are some rooms not coloured red, the player has failed, so exit via 3100.

3050 Print a suitable message using the subroutine at 6000.

3060–3070 If the player wishes to try the next level, continue at line 420.

3080 Otherwise, end the program via 7000.

3100–3110 If the player has missed some rooms, print a suitable message and allow the option of trying again at that level.

3120 Continue at line 460, for another try at the same level.

3130 Exit via 7000 if another try is not desired.

3500–3520 Print a message indicating failure; pause; and exit via 7000.

4000–4050 DATA for set attributes routine.

4100–4160 DATA for decode attributes routine.

5000–5030 DATA defining grid size and exit position, for 3 levels.

6000–6130 This subroutine is used to indicate success, and offer an attempt at the next level.

6200–6230 Detect a keypress.

6300–6370 This subroutine uses a pair of nested loops, and the decode attributes routine, to examine the PAPER colour in each room, and return with *r* set to the number of rooms which do not have red PAPER.

6400–6500 Exit via this routine when all levels have been successfully completed.

7000–7030 End the program.

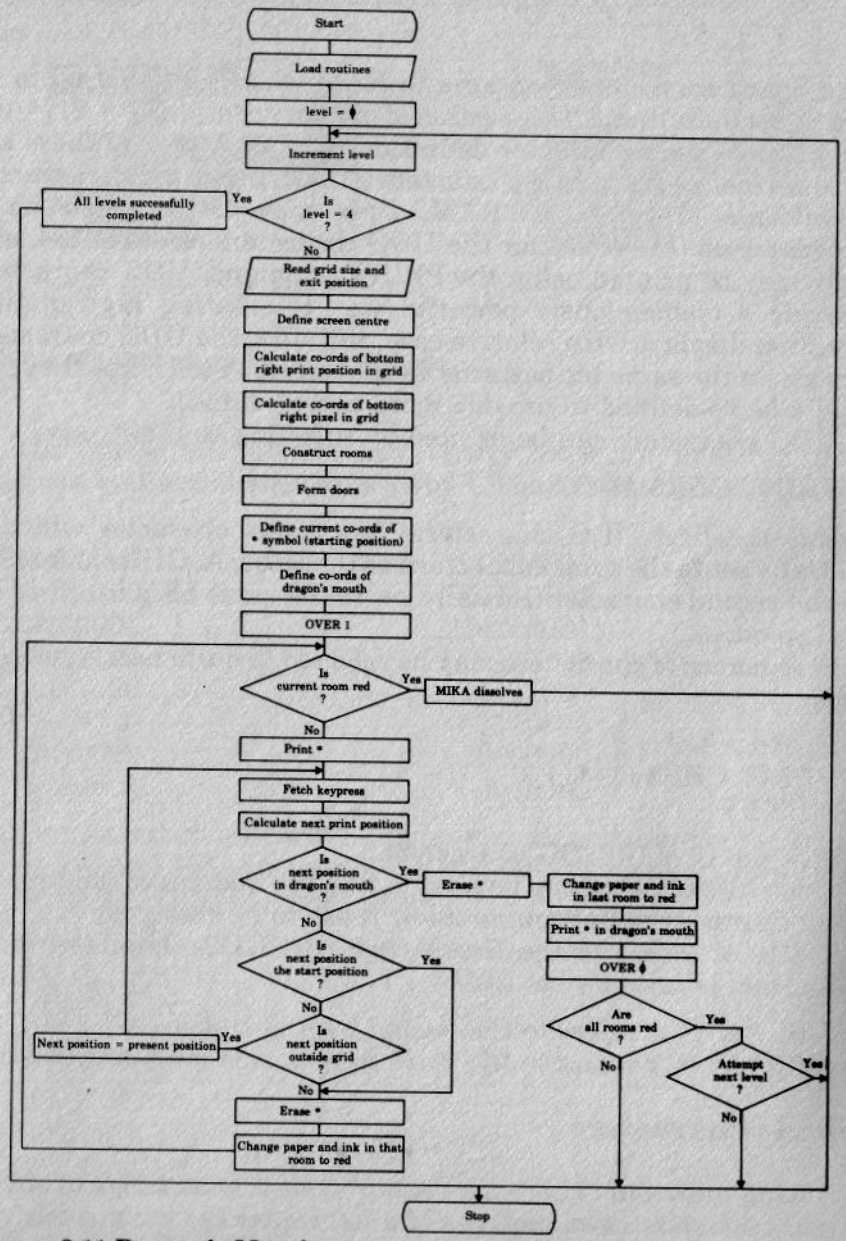


Figure 3.11 Dragon's Mouth

4 USER-DEFINED GRAPHICS

The Spectrum contains an area in RAM which can hold up to 21 character definitions. This section of memory contains 8 × 21 = 168 bytes, since each character definition occupies 8 bytes. These are the user-defined graphics characters (UDG). Because the character definitions are held in RAM, they may be changed by a programmer. The codes for the UDG characters are 144–164, and they may be printed using the PRINT command. UDG characters provide a tremendously powerful way of creating high quality graphics displays with relative ease. Initially, the UDG characters are given the same bit patterns as the letters A to U, but they can easily be re-defined to provide more useful shapes.

UDG characters can be printed by referring to their codes.

```
PRINT CHR$ 144
```

prints the first UDG character. This is the character which is initially set to the same bit pattern as the letter A. CHR$ 145 refers to the second character (initially set to the same bit pattern as B), and so on.

A sequence of characters may be referred to quite neatly, using a loop:

```
FOR i = 0 TO 9
PRINT CHR$ (144 + i)
NEXT i
```

prints the first 10 UDG characters.

The USR keyword can be used to find the address of the bytes in each character definition, to allow these to be changed.

USR "A" refers to the first byte of the UDG character whose definition is initially set to A.

USR "A" + 1 refers to the second byte in this area,
USR "A" + 2 refers to the third byte in this area, and so on.

Single characters

Printing individual characters can provide a wide range of effects. Figure 4.1 shows two pictures of a helicopter (go on, stretch your imagination!). First, the shapes are planned on an 8 × 8 grid. 1 mm

graph paper provides a good indication of the way the shape will appear on the screen, but larger squares are easier to work with once the outline has been fixed.

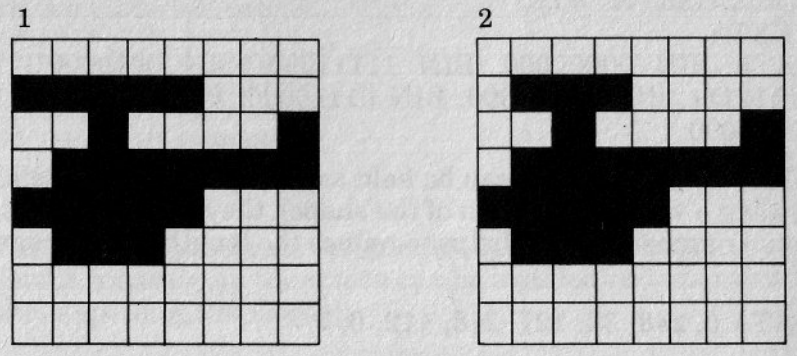


Figure 4.1 Two pictures of a helicopter

Turn these bit patterns into binary codes by replacing blank squares by 0, and shaded squares by 1. The corresponding codes are:

1	2
00000000	00000000
11111000	01110000
00100001	00100001
01111111	01111111
11111000	11111000
01110000	01110000
00000000	00000000
00000000	00000000

These character definitions can be placed in the areas with print codes 144 and 145 by using POKE and USR commands. This can be done using multiple statements:

```
POKE USR "A", BIN 00000000
POKE USR "A" +1, BIN 11111000
POKE USR "A" +2, BIN 00100001
POKE USR "A" +3, BIN 01111111
POKE USR "A" +4, BIN 11111000
POKE USR "A" +5, BIN 01110000
POKE USR "A" +6, BIN 00000000
POKE USR "A" +7, BIN 00000000
```

A neater way is to use a loop:

```
FOR i = 0 TO 7
READ c
POKE USR "A" +i,c
NEXT i
DATA BIN 00000000, BIN 11111000, BIN 00100001, BIN
01111111, BIN 11111000, BIN 01110000, BIN 00000000, BIN
00000000
```

While the bit patterns can be held as binary numbers (useful for providing a visual indication of the shape), they can also be held as decimal numbers, which helps to reduce the length of the program, that is:

```
DATA 0, 248, 33, 127, 248, 112, 0, 0
```

Having re-defined characters 144 and 145, repeatedly printing these on top of each other provides an illusion of movement.

Example 4.1

```
  10 REM HELICOPTER
  20 CLS
  30 PAPER 7: INK 0:  BORDER 7
  40 REM define UDG A
  50 FOR i=0 TO 7
  60 READ c
  70 POKE USR "A"+i,c
  80 NEXT i
  90 REM define UDG B
 100 FOR i=0 TO 7
 110 READ c
 120 POKE USR "B"+i,c
 130 NEXT i
 200 REM flying!
 210 FOR j=1 TO 50
 220 PRINT AT 10,16;CHR$ (144):
REM UDG A
 225 PAUSE 5
 230 PRINT AT 10,16;CHR$ (145):
REM UDG B
 235 PAUSE 5
 240 NEXT j
 500 REM data for UDG A
 510 DATA BIN 00000000
 520 DATA BIN 11111000
 530 DATA BIN 00100001
 540 DATA BIN 0i111111
 550 DATA BIN 11111000
 560 DATA BIN 01110000
```



```
 570 DATA BIN 00000000
 580 DATA BIN 00000000
 600 REM data for UDG B
 610 DATA BIN 00000000
 620 DATA BIN 01110000
 630 DATA BIN 00100001
 640 DATA BIN 01111111
 650 DATA BIN 11111000
 660 DATA BIN 01110000
 670 DATA BIN 00000000
 680 DATA BIN 00000000
```

Program 4.1

The printing process is sufficiently quick that shapes at several different positions on the screen can be handled one after the other, without spoiling the illusion.

Example 4.2

```
  10 REM HELICOPTERS
  20 CLS
  30 PAPER 7: INK 0:  BORDER 7
  40 REM define UDG A
  50 FOR i=0 TO 7
  60 READ c
  70 POKE USR "A"+i,c
  80 NEXT i
  90 REM define UDG B
 100 FOR i=0 TO 7
 110 READ c
 120 POKE USR "B"+i,c
 130 NEXT i
 200 REM flying!
 210 FOR j=1 TO 50
 220 PRINT AT 10,16;CHR$ (144):
REM UDG A
 222 PRINT AT 6,18;CHR$ (144)
 224 PRINT AT 8,17;CHR$ (144)
 226 PRINT AT 10,16;CHR$ (144)
 228 PRINT AT 12,17;CHR$ (144)
 230 PRINT AT 14,18;CHR$ (144)
 232 PRINT AT 6,18;CHR$ (145)
 234 PRINT AT 8,17;CHR$ (145)
 236 PRINT AT 10,16;CHR$ (145)
 238 PRINT AT 12,17;CHR$ (145)
 240 PRINT AT 14,18;CHR$ (145)
 242 NEXT j
 500 REM data for UDG A
 510 DATA BIN 00000000
 520 DATA BIN 11111000
 530 DATA BIN 00100001
 540 DATA BIN 01111111
```

Program 4.2 *(continues)*

```
550 DATA BIN 11111000
560 DATA BIN 01110000
570 DATA BIN 00000000
580 DATA BIN 00000000
600 REM data for UDG B
610 DATA BIN 00000000
620 DATA BIN 01110000
630 DATA BIN 00100001
640 DATA BIN 01111111
650 DATA BIN 11111000
660 DATA BIN 01110000
670 DATA BIN 00000000
680 DATA BIN 00000000
```

Program 4.2

Movement

Forward movement may be simulated by carrying out a series of print operations. Suppose that characters 144 and 145 hold representations of a shape in two positions as it moves.

(a) PRINT CHR$ 144 e.g., at 11,31
(b) PRINT CHR$ 32 over CHR$ 144, i.e., at 11,31
(c) PRINT CHR$ 145 in the next position to the left, i.e., at 11,30
(d) PRINT CHR$ 32 over CHR$ 145
(e) move one print position to the left, and repeat the steps from (a)

Example 4.3

```
 10 REM GNAT
 20 CLS
 30 PAPER 7: INK 0:  BORDER 7
 40 REM define UDG A
 50 FOR i=0 TO 7
 60 READ c
 70 POKE USR "A"+i,c
 80 NEXT i
 90 REM define UDG B
100 FOR i=0 TO 7
110 READ c
120 POKE USR "B"+i,c
130 NEXT i
200 REM flying!
210 FOR j=1 TO 10
220 LET x=31
230 LET y=11
240 PRINT AT y,x;CHR$ (144)
245 PAUSE 3
250 PRINT AT y,x;CHR$ (32)
260 LET x=x-1
```



```
270 PRINT AT y,x;CHR$ (145)
275 PAUSE 3
280 PRINT AT y,x;CHR$ (32)
290 LET x=x-1
300 REM check still on screen
310 REM next time
320 IF x>0 THEN  GO TO 240
330 NEXT j
500 REM data for UDG A
510 DATA BIN 00000000
520 DATA BIN 00000000
530 DATA BIN 00100000
540 DATA BIN 01111110
550 DATA BIN 00011100
560 DATA BIN 00011100
570 DATA BIN 00001000
580 DATA BIN 00000000
600 REM data for UDG B
610 DATA BIN 00000000
620 DATA BIN 00001000
630 DATA BIN 00111100
640 DATA BIN 01111110
650 DATA BIN 00000000
660 DATA BIN 00000000
670 DATA BIN 00000000
680 DATA BIN 00000000
```

Program 4.3

The printing-and-erasing technique produces movement, but it suffers from two principal faults; it is a little slow, and the movement is rather jerky. The printing process can be made more efficient by coupling each character with a SPACE character. Before beginning, define the values of two string variables as CHR$ 144 + CHR$ 32, and CHR$ 145 + CHR$ 32, that is,

LET a$ = CHR$ 144 + CHR$ 32
LET b$ = CHR$ 145 + CHR$ 32

Printing a$ now prints CHR$ 144 followed by a SPACE, performing tasks (d) and (a). Printing b$ performs tasks (b) and (c).

Example 4.4

```
10 REM GNAT 2
20 PAPER 7: INK 0: BORDER 7
30 CLS
40 REM define UDG A
50 FOR i=0 TO 7
60 READ c
70 POKE USR "A"+i,c
80 NEXT i
```

Program 4.4 *(continues)*

```
 90 REM define UDG B
100 FOR i=0 TO 7
110 READ c
120 POKE USR "B"+i,c
130 NEXT i
150 REM define character+space
160 LET a$=CHR$ 144+CHR$ 32
170 LET b$=CHR$ 145+CHR$ 32
200 REM flying!
210 FOR j=1 TO 10
220 LET x=31
230 LET y=11
240 PRINT AT y,x;a$
250 PAUSE 3
260 LET x=x-1
270 PRINT AT y,x;b$
280 PAUSE 3
290 LET x=x-1
300 REM check still on screen
310 REM next time
320 IF x>0 THEN  GO TO 240
330 PRINT AT y,x+1;CHR$ 32
340 NEXT j
500 REM data for UDG A
510 DATA BIN 00000000
520 DATA BIN 00000000
530 DATA BIN 00100000
540 DATA BIN 01111110
550 DATA BIN 00011100
560 DATA BIN 00011100
570 DATA BIN 00001000
580 DATA BIN 00000000
600 REM data for UDG B
610 DATA BIN 00000000
620 DATA BIN 00001000
630 DATA BIN 00111100
640 DATA BIN 01111110
650 DATA BIN 00000000
660 DATA BIN 00000000
670 DATA BIN 00000000
680 DATA BIN 00000000
```

Program 4.4

Line 325 is used to remove the gnat from the edge of the screen before beginning another pass. Try deleting this line, to see why it must be included.

The movement produced by printing is still rather jerky, as a result of the relatively large distance the object must move between printing positions. The CHARACTER PLOTTER routine from chapter 1 can be used to produce much smoother movement by

moving the characters 1 pixel (i.e., 1 bit) at a time rather than 1 printing position (i.e., 1 byte or 8 bits). In addition, the display process can be accelerated by paying attention to the original character design.

The present method uses a SPACE character to erase the current character before displaying the next character. However, the SPACE character is used to erase a whole PRINT position, and this is not necessary if the characters are only being moved 1 pixel at a time. All that is required is a border of unlit pixels around the character. Actually, a border of unlit pixels is only needed 'behind' the character ('behind' denoting the trailing edge of the character, relative to the direction in which it is travelling). Nevertheless, a border of unlit pixels around the character allows for movement in any direction, up, down, left or right.

Remember that the best way to handle the attributes when using the CHARACTER PLOTTER routine is to set them up in advance.

Example 4.5

```
 10 REM SMOOTH GNAT
 20 REM set colours
 30 PAPER 7: INK 0: BORDER 7
 40 CLS
100 REM set new ramtop
110 CLEAR 65164: REM 16K=32396
120 LET start=65165: REM 16K=32
397
130 LET s=start
140 LET length=203
150 FOR i=1 TO length
160 READ n
170 POKE s,n
180 LET s=s+1
190 NEXT i
200 REM define UDG A
210 FOR i=0 TO 7
220 READ c
230 POKE USR "A"+i,c
240 NEXT i
250 REM define UDG B
260 FOR i=0 TO 7
270 READ c
280 POKE USR "B"+i,c
290 NEXT i
300 REM flying again !
310 FOR j=1 TO 10
320 LET x=248
330 LET y=80
340 POKE 23296,x: POKE 23297,y:
POKE 23298,144
```

Program 4.5 *(continues)*

```
 350 RANDOMIZE USR start
 360 LET x=x-1
 370 POKE 23296,x: POKE 23297,y:
 POKE 23298,145
 380 RANDOMIZE USR start
 390 LET x=x-1
 400 IF x>0 THEN  GO TO 340
 410 LET x=x+1
 420 POKE 23296,x: POKE 23297,y:
 POKE 23298,32
 430 RANDOMIZE USR start
 440 NEXT j
1000 REM CHARACTER PLOTTER
1010 DATA 58,2,91,254,32,56,76,2
54,128,56
1020 DATA 10,254,144,56,68,254,1
65,56,9,24
1030 DATA 62,33,7,61,214,32,24,6
,42,123
1040 DATA 92,43,214,143,17,8,0,2
54,0,40
1050 DATA 4,25,61,24,248,6,0,126
,43,229
1060 DATA 14,0,24,1,241,7,245,19
7,245,33
1070 DATA 0,91,126,254,249,48,18
,129,79,35
1080 DATA 126,254,169,48,10,128,
71,24,8,24
1090 DATA 229,24,220,24,117,24,1
11,62,16,128
1100 DATA 71,33,0,64,254,128,48,
9,17,0
1110 DATA 8,25,254,64,48,1,25,20
3,184,203
1120 DATA 176,62,63,144,17,32,0,
254,8,56
1130 DATA 5,214,8,25,24,247,254,
0,40,4
1140 DATA 61,36,24,248,121,254,8
,56,5,214
1150 DATA 8,35,24,247,79,62,7,14
5,55,63
1160 DATA 87,71,94,254,0,40,4,20
3,11,16
1170 DATA 252,241,56,4,203,131,2
4,2,203,195
1180 DATA 122,66,254,0,40,4,203,
3,16,252
1190 DATA 115,193,12,121,254,8,3
2,147,4,241
1200 DATA 225,120,254,8,32,141,4
0,4,241,193
1210 DATA 241,225,201
2000 REM define UDG A
```



```
2010 DATA BIN 00000000
2020 DATA BIN 00000000
2030 DATA BIN 00100000
2040 DATA BIN 01111110
2050 DATA BIN 00011100
2060 DATA BIN 00011100
2070 DATA BIN 00001000
2080 DATA BIN 00000000
2100 REM define UDG B
2110 DATA BIN 00000000
2120 DATA BIN 00001000
2130 DATA BIN 00111100
2140 DATA BIN 01111110
2150 DATA BIN 00000000
2160 DATA BIN 00000000
2170 DATA BIN 00000000
2180 DATA BIN 00000000
```

Program 4.5

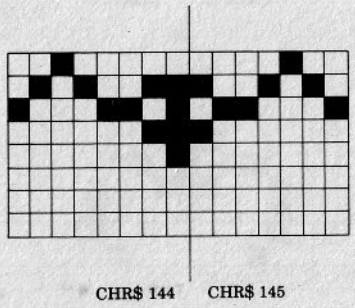
Multiple characters

The shapes which can be constructed inside a single UDG character are rather limited in size, especially if there is a border of unlit pixels around the shape to allow movement.

Fortunately, there are 21 possible UDG characters, so a larger shape may be constructed from more than 1 character.

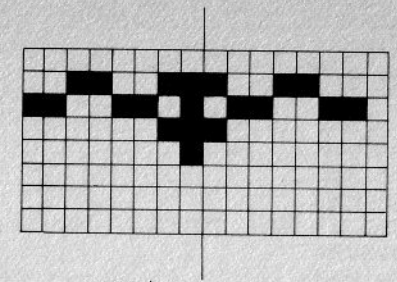
SOLITARY SEAGULL

The illusion of a seagull flying can be produced using 5 shapes, each employing 2 UDG characters, as shown in Fig. 4.2 a–e. The same illusion could be produced using fewer shapes, but this seagull is a really stylish aeronaut!



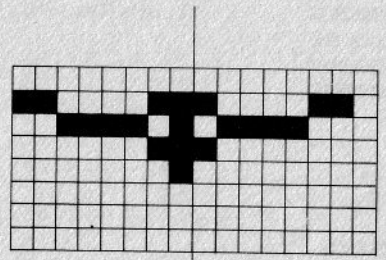
(a)

Figure 4.2 *(continues)*



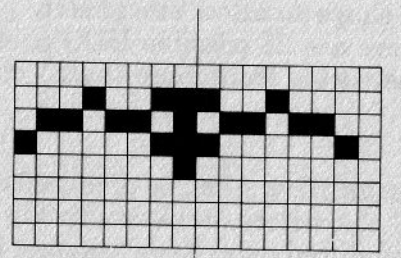
CHR$ 146 CHR$ 147

(b)



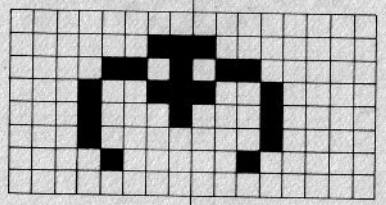
CHR$ 148 CHR$ 149

(c)



CHR$ 150 CHR$ 151

(d)



CHR$ 152 CHR$ 153

(e)

Figure 4.2

The following example shows the seagull flying.
Each shape is produced in one PRINT operation by defining

b$(1) = CHR$ 144 + CHR$ 145 shape 1
b$(2) = CHR$ 146 + CHR$ 147 shape 2
b$(3) = CHR$ 148 + CHR$ 149 shape 3
b$(4) = CHR$ 150 + CHR$ 151 shape 4
b$(5) = CHR$ 152 + CHR$ 153 shape 5

A FOR . . . NEXT loop is used to print the shapes repeatedly, in order.

Example 4.6

```
 100 REM BIRD FLYING
 110 CLS
 120 DIM b$(5,2)
 200 REM Define print strings
 210 LET b$(1)=CHR$ (144)+CHR$ (
145)
 220 LET b$(2)=CHR$ (146)+CHR$ (
147)
 230 LET b$(3)=CHR$ (148)+CHR$ (
149)
 240 LET b$(4)=CHR$ (150)+CHR$ (
151)
 250 LET b$(5)=CHR$ (152)+CHR$ (
153)
 300 REM Define characters
 310 REM i steps through charact
ers
 320 REM j steps through rows
 330 FOR i=1 TO 10
 340 READ c$
 350 FOR j=0 TO 7
 360 READ d
 370 POKE USR c$+j,d
 380 NEXT j
 390 NEXT i
 400 REM Draw flying bird
 410 REM i is total number of fl
aps
 420 REM j steps through positio
ns
 430 FOR i=1 TO 20
 440 FOR j=1 TO 5
 450 PRINT AT 11,16;b$(j)
 455 PAUSE 3
 460 NEXT j
 470 FOR j=5 TO 1 STEP -1
 480 PRINT AT 11,16;b$(j)
 485 PAUSE 3
```

Program 4.6 *(continues)*

```
 490 NEXT j
 500 NEXT i
1000 REM UDG codes
1010 DATA "a"
1020 DATA BIN 00100000
1030 DATA BIN 01010011
1040 DATA BIN 10001101
1050 DATA BIN 00000011
1060 DATA BIN 00000001
1070 DATA BIN 00000000
1080 DATA BIN 00000000
1090 DATA BIN 00000000
1100 DATA "b"
1110 DATA BIN 00001000
1120 DATA BIN 10010100
1130 DATA BIN 01100010
1140 DATA BIN 10000000
1150 DATA BIN 00000000
1160 DATA BIN 00000000
1170 DATA BIN 00000000
1180 DATA BIN 00000000
1200 DATA "c"
1210 DATA BIN 00000000
1220 DATA BIN 00110011
1230 DATA BIN 11001101
1240 DATA BIN 00000011
1250 DATA BIN 00000001
1260 DATA BIN 00000000
1270 DATA BIN 00000000
1280 DATA BIN 00000000
1300 DATA "d"
1310 DATA BIN 00000000
1320 DATA BIN 10011000
1330 DATA BIN 01100110
1340 DATA BIN 10000000
1350 DATA BIN 00000000
1360 DATA BIN 00000000
1370 DATA BIN 00000000
1380 DATA BIN 00000000
1400 DATA "e"
1410 DATA BIN 00000000
1420 DATA BIN 11000011
1430 DATA BIN 00111101
1440 DATA BIN 00000011
1450 DATA BIN 00000001
1460 DATA BIN 00000000
1470 DATA BIN 00000000
1480 DATA BIN 00000000
1500 DATA "f"
1510 DATA BIN 00000000
1520 DATA BIN 10000110
1530 DATA BIN 01111000
1540 DATA BIN 10000000
1550 DATA BIN 00000000
1560 DATA BIN 00000000
```



```
1570 DATA BIN 00000000
1580 DATA BIN 00000000
1600 DATA "g"
1610 DATA BIN 00000000
1620 DATA BIN 00010011
1630 DATA BIN 01101101
1640 DATA BIN 10000011
1650 DATA BIN 00000001
1660 DATA BIN 00000000
1670 DATA BIN 00000000
1680 DATA BIN 00000000
1700 DATA "h"
1710 DATA BIN 00000000
1720 DATA BIN 10010000
1730 DATA BIN 01101100
1740 DATA BIN 10000010
1750 DATA BIN 00000000
1760 DATA BIN 00000000
1770 DATA BIN 00000000
1780 DATA BIN 00000000
1800 DATA "i"
1810 DATA BIN 00000000
1820 DATA BIN 00000011
1830 DATA BIN 00001101
1840 DATA BIN 00010011
1850 DATA BIN 00010001
1860 DATA BIN 00010000
1870 DATA BIN 00001000
1880 DATA BIN 00000000
1900 DATA "j"
1910 DATA BIN 00000000
1920 DATA BIN 10000000
1930 DATA BIN 01100000
1940 DATA BIN 10010000
1950 DATA BIN 00010000
1960 DATA BIN 00010000
1970 DATA BIN 00100000
1980 DATA BIN 00000000
```

Program 4.6

Varying the PAUSE in lines 455 and 485 varies the speed of flight. An increase in speed is also possible if as many constants as possible are replaced by variables previously defined to the same values as the constants. There is not much scope for improvement in this particular program, but even replacing the initial and final values in the FOR . . . NEXT loop is worthwhile.

SYNCHRONOUS SEAGULLS

Larger numbers of UDG characters may be printed, although this naturally tends to slow the action down.

Example 4.7

Note the reduction in the space occupied by the DATA statements, simply by turning the UDG definitions into decimal numbers.

```
 100 REM BIRDS FLYING
 110 CLS
 120 DIM b$(5,2)
 200 REM Define print strings
 210 LET b$(1)=CHR$ (144)+CHR$ (
145)
 220 LET b$(2)=CHR$ (146)+CHR$ (
147)
 230 LET b$(3)=CHR$ (148)+CHR$ (
149)
 240 LET b$(4)=CHR$ (150)+CHR$ (
151)
 250 LET b$(5)=CHR$ (152)+CHR$ (
153)
 300 REM Define characters
 310 REM i steps through charact
ers
 320 REM j steps through rows
 330 FOR i=1 TO 10
 340 READ c$
 350 FOR j=0 TO 7
 360 READ d
 370 POKE USR c$+j,d
 380 NEXT j
 390 NEXT i
 400 REM draw flying birds
 410 REM i is total number of fl
aps
 420 REM j steps through positio
ns
 430 FOR i=1 TO 20
 440 FOR j=1 TO 5
 445 FOR x=12 TO 18 STEP 2
 450 PRINT AT 11,x;b$(j)
 455 NEXT x
 460 NEXT j
 470 FOR j=5 TO 1 STEP -1
 475 FOR x=12 TO 18 STEP 2
 480 PRINT AT 11,x;b$(j)
 485 NEXT x
 490 NEXT j
 500 NEXT i
1000 REM UDG codes
1010 DATA "a"
1020 DATA 32,83,149,3,1,0,0,0
1100 DATA "b"
1110 DATA 8,148,98,128,0,0,0,0
1200 DATA "c"
```



```
1210 DATA 0,51,213,3,1,0,0,0
1300 DATA "d"
1310 DATA 0,152,102,128,0,0,0,0
1400 DATA "e"
1410 DATA 0,195,125,3,1,0,0,0
1500 DATA "f"
1510 DATA 0,134,120,128,0,0,0,0
1600 DATA "g"
1610 DATA 0,19,109,131,1,0,0,0
1700 DATA "h"
1710 DATA 0,144,108,130,0,0,0,0
1800 DATA "i"
1810 DATA 0,3,13,19,17,16,8,0
1900 DATA "j"
1910 DATA 0,128,96,144,16,16,32,
0
```

Program 4.7

Part of the problem in this program is that all those loops take time. To increase the speed a little, define larger print strings, for example:

LET b$(1) = CHR$ 144 + CHR$ 145 + CHR$ 144 + CHR$ 145 + CHR$ 144 + CHR$ 145 + CHR$ 144 + CHR$ 145

This will allow a single PRINT statement to be used (as in Example 4.6). Replacing the line and column values in the PRINT statements with variables helps too.

Shapes constructed using larger numbers of UDG characters need some attention to detail.

There are 8 separate UDG characters in the shape shown in Fig. 4.3.

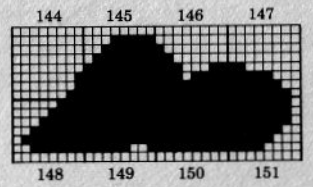


Figure 4.3 A cloud

Printing the individual characters is best done using line and column values which are related to one point on the shape, e.g., the left-hand corner. This point is not picked at random – it was chosen because it allows the character plotter routine to be used as well as the PRINT command. Figure 4.4 shows the effect of using x to represent the column, and y to represent the line of the bottom left-hand corner of each row of characters.

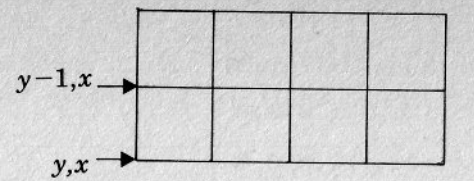


Figure 4.4 The co-ordinates of the bottom left corner of each row

At the beginning of the program, assign the two rows of characters to a string, for example:

```
LET a$ = CHR$ 144 + CHR$ 145 + CHR$ 146 + CHR$ 147 +
CHR$ 32
LET b$ = CHR$ 148 + CHR$ 149 + CHR$ 150 + CHR$ 151 +
CHR$ 32
```

The shape may then be printed using

```
PRINT AT y – 1,x;a$
PRINT AT y,x;b$
```

and the usual technique of advancing and printing again may be used, to produce movement, since each string has a SPACE character at the end.

Example 4.8

```
 100 REM CLOUD
 110 REM set colours
 120 PAPER 1: INK 7: BORDER 1
 130 CLS
 200 REM define UDGs
 210 FOR j=1 TO 8: REM count cha
racters
 220 READ c$: REM character
 230 FOR i=0 TO 7: REM character
 bytes
 240 READ c
 250 POKE USR c$+i,c
 260 NEXT i
 270 NEXT j
 300 REM define print strings
 310 LET a$=CHR$ 144+CHR$ 145+CH
R$ 146+CHR$ 147+CHR$ 32
 320 LET b$=CHR$ 148+CHR$ 149+CH
R$ 150+CHR$ 151+CHR$ 32
 400 REM move cloud
 410 LET y=5: REM height
 420 FOR j=1 TO 10: REM no. of t
imes across screen
```



```
 430 FOR x=27 TO 0 STEP -1: REM
front of cloud
 440 PRINT AT y-1,x;a$
 450 PRINT AT y,x;b$
 460 PAUSE 3
 470 NEXT x
 480 PRINT AT y-1,0;"     ": REM
erase top of cloud
 490 PRINT AT y,0;"     ": REM er
ase bottom of cloud
 500 NEXT j
1000 REM UDG definitions
1010 DATA "A",0,0,0,0,0,1,1,3
1020 DATA "B",0,31,63,127,255,25
5,255,255
1030 DATA "C",0,0,128,192,227,23
9,255,255
1040 DATA "D",0,0,0,0,192,248,25
2,254
1050 DATA "E",7,15,31,63,127,127
,63,0
1060 DATA "F",255,255,255,255,25
5,255,249,0
1070 DATA "G",255,255,255,255,25
5,255,255,0
1080 DATA "H",254,254,254,252,24
8,240,0,0
```

Program 4.8

Since clouds are normally supposed to move smoothly and gracefully across the sky, the CHARACTER PLOTTER routine can be used to produce the shape. Notice that the shape was originally defined with a row of unlit pixels round the outline, to allow for movement in steps of 1 pixel. If the CHARACTER PLOTTER routine is to be used, the position of the left-hand corner of each character in the shape will be as shown in Fig. 4.5.

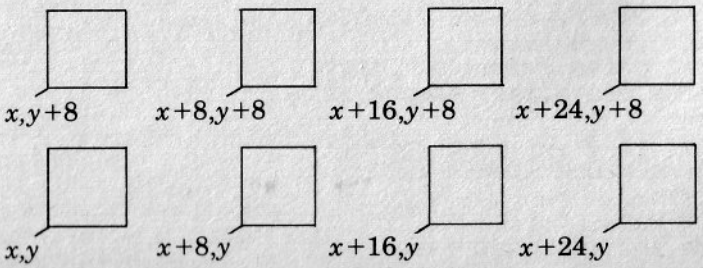


Figure 4.5 Bottom left PLOT co-ordinates

Example 4.9

```
  10 REM SMOOTH CLOUD
  20 REM set colours
  30 PAPER 1: INK 7: BORDER 1
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65164: REM 16K=32396
 120 LET start=65165: REM 16K=32
397
 200 REM place routine
 210 LET length=203
 220 LET s=start
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 300 REM define UDGs
 310 FOR j=1 TO 8
 320 READ c$
 330 FOR i=0 TO 7
 340 READ c
 350 POKE USR c$+i,c
 360 NEXT i
 370 NEXT j
 400 REM move cloud
 410 LET y=136
 420 FOR j=1 TO 3
 430 FOR x=224 TO 0 STEP -1
 440 POKE 23296,x
 450 POKE 23297,y+8: POKE 23298,
144: RANDOMIZE USR start
 460 POKE 23297,y: POKE 23298,14
8: RANDOMIZE USR start
 470 POKE 23296,x+8: POKE 23297,
y+8: POKE 23298,145: RANDOMIZE U
SR start
 480 POKE 23297,y: POKE 23298,14
9: RANDOMIZE USR start
 490 POKE 23296,x+16: POKE 23297
,y+8: POKE 23298,146: RANDOMIZE
USR start
 500 POKE 23297,y: POKE 23298,15
0: RANDOMIZE USR start
 510 POKE 23296,x+24: POKE 23297
,y+8: POKE 23298,147: RANDOMIZE
USR start
 520 POKE 23297,y: POKE 23298,15
1: RANDOMIZE USR start
 530 NEXT x
 540 REM delete cloud
 550 FOR x=0 TO 24 STEP 8
 560 POKE 23296,x: POKE 23297,y+
8: POKE 23298,32: RANDOMIZE USR
```

Program 4.9 *(continues)*


```
start
 570 POKE 23297,y: RANDOMIZE USR
 start
 580 NEXT x
 590 NEXT j
 600 REM finished
 610 REM restore colours
 620 PAPER 7: INK 0: BORDER 7
 630 CLS
1000 REM CHARACTER PLOTTER
1010 DATA 58,2,91,254,32,56,76,2
54,128,56
1020 DATA 10,254,144,56,68,254,1
65,56,9,24
1030 DATA 62,33,7,61,214,32,24,6
,42,123
1040 DATA 92,43,214,143,17,8,0,2
54,0,40
1050 DATA 4,25,61,24,248,6,0,126
,43,229
1060 DATA 14,0,24,1,241,7,245,19
7,245,33
1070 DATA 0,91,126,254,249,48,18
,129,79,35
1080 DATA 126,254,169,48,10,128,
71,24,8,24
1090 DATA 229,24,220,24,117,24,1
11,62,16,128
1100 DATA 71,33,0,64,254,128,48,
9,17,0
1110 DATA 8,25,254,64,48,1,25,20
3,184,203
1120 DATA 176,62,63,144,17,32,0,
254,8,56
1130 DATA 5,214,8,25,24,247,254,
0,40,4
1140 DATA 61,36,24,248,121,254,8
,56,5,214
1150 DATA 8,35,24,247,79,62,7,14
5,55,63
1160 DATA 87,71,94,254,0,40,4,20
3,11,16
1170 DATA 252,241,56,4,203,131,2
4,2,203,195
1180 DATA 122,66,254,0,40,4,203,
3,16,252
1190 DATA 115,193,12,121,254,8,3
2,147,4,241
1200 DATA 225,120,254,8,32,141,4
0,4,241,193
1210 DATA 241,225,201
2000 REM UDG DEFINITIONS
2010 DATA "A",0,0,0,0,0,1,1,3
2020 DATA "B",0,31,63,127,255,25
5,255,255
```

```
2030 DATA "C",0,0,128,192,227,23
9,255,255
2040 DATA "D",0,0,0,0,192,248,25
2,254
2050 DATA "E",7,15,31,63,127,127
,63,0
2060 DATA "F",255,255,255,255,25
5,255,249,0
2070 DATA "G",255,255,255,255,25
5,255,255,0
2080 DATA "H",254,254,254,252,24
8,240,0,0
```

Program 4.9

With a large shape such as this, it is worth experimenting with the order in which the individual characters are printed. Printing the characters in the order shown in Fig. 4.6 was fine when the PRINT statement was used.

1	2	3	4	5
6	7	8	9	10

Figure 4.6 PRINTing order (5 and 10 are 'space' characters)

Printing the characters in a different order, as shown in Fig. 4.7, produces a better illusion when the CHARACTER PLOTTER routine is used.

1	3	5	7
2	4	6	8

Figure 4.7 PLOTting order

This improvement results because the new order of printing produces less apparent distortion at the front of the cloud, for a shorter time, than the previous order of printing. The effect now is to 'stretch' the cloud slightly, which is more acceptable than moving the whole of the top half, then the whole of the bottom half.

Inserting PAUSE statements can have an interesting effect, for example,

```
525 PAUSE 10
```

Much of the remaining problem of jerk is caused by the frequent returns from machine code to BASIC and back again. This can be greatly reduced by defining the addresses 23296–23298 as variables, along with the use of long lines of multiple statements. Note that it is not necessary to set all the values of *x*, *y*, and CODE each time a character is to be plotted. Usually, at least one of the previously set values can be retained.

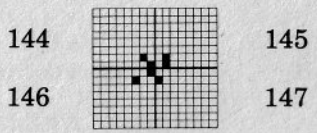
Explosions

An explosion occupies a small area at first, but quickly expands, changing shape as it does so. It then disappears, revealing
(a) an object which looks as though it has been blown to bits, or
(b) a background scene in which there is no trace of the object which has just exploded.

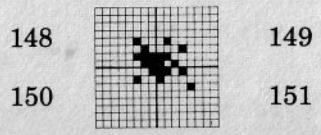
UDG characters provide an ideal way of simulating explosions. Define sufficient characters to represent the various stages of the explosion and display these, singly or in groups, one after the other. The OVER1 command is particularly useful here, because in an explosion there are usually fragments flying through the air, and the background will normally be visible behind these.

Example 4.10

Figure 4.8a–d shows the explosion (the character codes are printed at the sides).

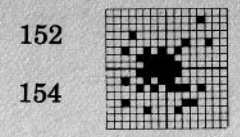


(a)

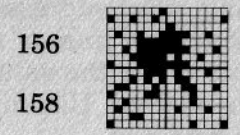


(b)

Figure 4.8 *(continues)*



(c)



(d)

Figure 4.8

The demonstration involves a helicopter, printed at 11,16. The explosion takes place a little in front of, and slightly below, the helicopter, and occupies print positions 11,15 11,16 12,15 and 12,16. The characters forming the explosion as it expands are printed using OVER1, although they quickly obscure most of the helicopter. The same characters are printed in reverse order as the explosion fades away. This time, OVER0 is used, so that the helicopter is removed as the explosion contracts. Finally, the last vestiges of the explosion are wiped away. Removing the PAUSE statements produces a very fast explosion.

```
 10 REM EXPLODING HELICOPTER
 20 REM set colours
 30 PAPER 5: INK 0: BORDER 5
 40 CLS
100 REM define UDGs
110 FOR i=1 TO 18
120 READ c$
130 FOR j=0 TO 7
140 READ d
150 POKE USR c$+j,d
160 NEXT j
170 NEXT i
200 REM fly helicopter
210 FOR i=1 TO 20
220 PRINT AT 11,16;CHR$ 144
230 PAUSE 5
240 PRINT AT 11,16;CHR$ 145
250 PAUSE 5
260 NEXT i
300 REM explosion expanding
310 OVER 1
320 PRINT AT 11,15;CHR$ 146+CHR
```



```
$ 147
 330 PRINT AT 12,15;CHR$ 148+CHR
$ 149
 335 PAUSE 3
 340 PRINT AT 11,15;CHR$ 150+CHR
$ 151
 350 PRINT AT 12,15;CHR$ 152+CHR
$ 153
 355 PAUSE 3
 360 PRINT AT 11,15;CHR$ 154+CHR
$ 155
 370 PRINT AT 12,15;CHR$ 156+CHR
$ 157
 375 PAUSE 3
 380 PRINT AT 11,15;CHR$ 158+CHR
$ 159
 390 PRINT AT 12,15;CHR$ 160+CHR
$ 161
 395 PAUSE 3
 400 REM explosion contracting
 410 OVER 0
 420 PRINT AT 11,15;CHR$ 158+CHR
$ 159
 430 PRINT AT 12,15;CHR$ 160+CHR
$ 161
 435 PAUSE 3
 440 PRINT AT 11,15;CHR$ 154+CHR
$ 155
 450 PRINT AT 12,15;CHR$ 156+CHR
$ 157
 455 PAUSE 3
 460 PRINT AT 11,15;CHR$ 150+CHR
$ 151
 470 PRINT AT 12,15;CHR$ 152+CHR
$ 153
 475 PAUSE 3
 480 PRINT AT 11,15;CHR$ 146+CHR
$ 147
 490 PRINT AT 12,15;CHR$ 148+CHR
$ 149
 495 PAUSE 3
 500 REM remove explosion
 510 PRINT AT 11,15;CHR$ 32+CHR$
 32
 520 PRINT AT 12,15;CHR$ 32+CHR$
 32
 600 REM finished
 610 PAPER 7: INK 0: BORDER 7
 620 CLS
1000 REM UDG DATA
1010 DATA "A",0,248,33,127,248,1
12,0,0
1020 DATA "B",0,112,33,127,248,1
12,0,0
1030 DATA "C",0,0,0,0,0,0,2,1
```

Program 4.10 *(continues)*

```
1040 DATA "D",0,0,0,0,0,0,64,64
1050 DATA "E",1,4,0,0,0,0,0,0
1060 DATA "F",0,64,0,0,0,0,0,0
1070 DATA "G",0,0,0,0,4,2,7,3
1080 DATA "H",0,0,0,0,64,16,192,
160
1090 DATA "I",1,4,0,0,0,0,0,0
1100 DATA "J",208,128,8,0,0,0,0,
0
1110 DATA "K",0,0,0,16,0,37,15,7
1120 DATA "L",0,8,0,0,36,64,128,
192
1130 DATA "M",15,37,10,0,32,4,0,
0
1140 DATA "N",192,128,232,0,48,6
4,0,0
1150 DATA "O",1,146,0,73,15,47,3
1,143
1160 DATA "P",0,1,128,228,192,22
6,72,224
1170 DATA "Q",7,21,98,134,32,64,
12,128
1180 DATA "R",68,96,50,16,16,68,
0,17
```

Program 4.10

An improvement might be to make the helicopter fly across the screen; produce the explosion; then move the explosion down the screen as it contracts, to simulate the fragments of the helicopter falling from the sky.

Foreground and background

Careful choice of PAPER and INK colours can help simulate depth in a picture. The following example shows a van driving past a house. First, the van drives in front of the house. Then the van driver discovers he has forgotten his lunch, so he reverses, driving behind the house on his way past.

When the house is created, the ground floor is formed from two SPACE characters printed with PAPER 6 and INK 0. Any object drawn in either of the print positions forming the ground floor will appear black. Since the SPACE character does not contain any lit pixels, the ground has a uniform colour – the PAPER colour. The van is plotted using the CHARACTER PLOTTER routine which turns pixel ON (in the INK colour of that position) and OFF (when the pixels revert to the PAPER colour). So, as the van passes the ground floor for the first time, lit pixels appear black, and unlit

pixels appear yellow. The van, therefore, is 'visible' as it passes 'in front of' the house.

Before the van reverses past the house, the ground floor is re-printed, using PAPER 6 and INK 6. Now, any lit pixels on the ground floor will not be seen, because they will be the same colour as the PAPER. When reversing, the van is drawn 'invisibly' as it passes the house, and so it appears to pass 'behind' the house.

Example 4.11

```
 10 REM MOVING PAST
 20 REM set colours
 30 PAPER 5: INK 0: BORDER 5
 40 CLS
100 REM set new ramtop
110 CLEAR 65164: REM 16k=32396
120 LET start=65165: REM 16K=32
397
130 LET s=start
140 LET length=203
150 FOR i=1 TO length
160 READ n
170 POKE s,n
180 LET s=s+1
190 NEXT i
200 REM define UDGs
210 FOR i=1 TO 4
220 READ c$
230 FOR j=0 TO 7
240 READ d
250 POKE USR c$+j,d
260 NEXT j
270 NEXT i
300 REM draw ground
310 PAPER 4
320 FOR l=12 TO 21
330 FOR c=0 TO 31
340 PRINT AT l,c;CHR$ 32
350 NEXT c
360 NEXT l
400 REM place house
410 INK 0: PAPER 6
420 PRINT AT 11,15;CHR$ 32+CHR$
32
430 INK 2: PAPER 5
440 PRINT AT 10,15;CHR$ 144+CHR
$ 145
500 REM move in front of house
510 LET y=80
520 POKE 23297,y
530 FOR x=0 TO 192
540 POKE 23296,x+8: POKE 23298,
```

Program 4.11 *(continues)*

```
146: RANDOMIZE USR start
 550 POKE 23296,x: POKE 23298,14
7: RANDOMIZE USR start
 560 NEXT x
 600 REM change INK on bottom
 610 REM of house
 620 INK 6: PAPER 6
 630 PRINT AT 11,15;CHR$ 32+CHR$
32
 700 REM move behind house
 710 FOR x=200 TO 8 STEP -1
 720 POKE 23296,x-8: POKE 23298,
147: RANDOMIZE USR start
 730 POKE 23296,x: POKE 23298,14
6: RANDOMIZE USR start
 740 NEXT x
1000 REM CHARACTER PLOTTER
1010 DATA 58,2,91,254,32,56,76,2
54,128,56
1020 DATA 10,254,144,56,68,254,1
65,56,9,24
1030 DATA 62,33,7,61,214,32,24,6
,42,123
1040 DATA 92,43,214,143,17,8,0,2
54,0,40
1050 DATA 4,25,61,24,248,6,0,126
,43,229
1060 DATA 14,0,24,1,241,7,245,19
7,245,33
1070 DATA 0,91,126,254,249,48,18
,129,79,35
1080 DATA 126,254,169,48,10,128,
71,24,8,24
1090 DATA 229,24,220,24,117,24,1
11,62,16,128
1100 DATA 71,33,0,64,254,128,48,
9,17,0
1110 DATA 8,25,254,64,48,1,25,20
3,184,203
1120 DATA 176,62,63,144,17,32,0,
254,8,56
1130 DATA 5,214,8,25,24,247,254,
0,40,4
1140 DATA 61,36,24,248,121,254,8
,56,5,214
1150 DATA 8,35,24,247,79,62,7,14
5,55,63
1160 DATA 87,71,94,254,0,40,4,20
3,11,16
1170 DATA 252,241,56,4,203,131,2
4,2,203,195
1180 DATA 122,66,254,0,40,4,203,
3,16,252
1190 DATA 115,193,12,121,254,8,3
2,147,4,241
```



```
1200 DATA 225,120,254,8,32,141,4
0,4,241,193
1210 DATA 241,225,201
2000 REM UDG DATA
2010 DATA "A",1,3,7,63,63,63,127
,255
2020 DATA "B",128,192,226,246,25
4,254,254,255
2030 DATA "C",240,240,144,144,25
4,254,254,48
2040 DATA "D",127,127,127,127,12
7,127,127,48
```

Program 4.11

Additional UDG characters

Sometimes, there may be a need for more than the 21 UDG characters the Spectrum can normally handle. Fortunately, it is possible to extend the number of characters by setting aside an additional area in memory for the character definitions. Extra characters whose definitions are held in this area may be printed using the PRINT command or the CHARACTER PLOTTER, SCALED CHARACTER PLOTTER or ROTATED SCALED CHARACTER PLOTTER routines.

The Spectrum keeps track of where the UDG definitions are stored by setting the UDG system variable to the address of the start of the first character definition. By varying the address held in the system variable the character set which is used may be altered, e.g., the character set may be reduced in size to allow more program or workspace.

However, it is much more useful to be able to extend the UDG character set. The problem here is that the characters used when defining the UDG character set are restricted to 'A' to 'U', that is,

```
POKE USR "U"+7,c
```

refers to the last byte in the normal UDG character set. Despite this apparent restriction, it is easy to create a second, additional, set of UDG characters and to set the system variable to point either to the original set or to the additional set. The printer buffer has sufficient space available to be able to hold a complete set of 21 alternate character definitions. In this case, it is wise to avoid the first few locations, to leave workspace for some of the machine code routines. If the address of the start of the alternate set of definitions is set to 23383, that will allow 88 locations workspace (more than enough) plus 168 (i.e., 21 × 8) locations for up to 21 alternate character definitions.

The UDG system variable occupies locations 23675 and 23676. Setting this system variable involves placing the address of the start of the UDG character set in 23675 (low bit) and 23676 (high bit). This can be accomplished using BASIC statements. Suppose that the address 23383 is to be the start of the character definitions:

```
LET address = 23383
POKE 23675,address–256*INT(address/256)
POKE 23676,INT(address/256)
```

will set the UDG system variable to point to the alternate character set, while

(48K)

```
LET address=65368
POKE 23675,address–256*INT(address/256)
POKE 23676,INT(address/256)
```

(16K)

```
LET address=32600
POKE 23675,address–256*INT(address/256)
POKE 23676,INT(address/256)
```

changes the system variable to point to the original UDG character set once again.

Of course the printer buffer does not have to be used. The alternate character set could be placed above RAMTOP, provided the system variable is set to a suitable value.

The following example shows how to place a character definition in the original UDG area, using USR "A", and a different definition in a new area (in the printer buffer, starting at 23383), also using USR "A". These two characters are printed using CHR$ 144 to refer to each character, by first setting the UDG system variable to point to the appropriate character set.

Example 4.12

```
 10 REM GREENIE
 20 PAPER 7: INK 4: BORDER 7
 30 CLS
100 REM first definition
110 FOR i=0 TO 7
120 READ c
130 POKE USR "a"+i,c
140 NEXT i
200 REM second definition
210 REM alternative system vari
able value
220 LET address=23383: POKE 236
75,address-256*INT (address/256)
: POKE 23676,INT (address/256)
230 REM load data
240 FOR i=0 TO 7
250 READ c
260 POKE USR "a"+i,c
270 NEXT i
300 REM display characters
310 FOR j=1 TO 50: REM no. of r
epetitions
320 REM original system variabl
e value
330 LET address=65368: POKE 236
75,address-256*INT (address/256)
: POKE 23676,INT (address/256):
REM 16K address=32600
335 PAUSE 10
340 PRINT AT 11,16;CHR$ (144)
350 REM change system variable
to alternative value
360 LET address=23383: POKE 236
75,address-256*INT (address/256)
: POKE 23676,INT (address/256)
365 PAUSE 10
370 PRINT AT 11,16;CHR$ (144)
380 NEXT j
400 REM reset colours
410 PAPER 7: INK 0 : BORDER 7
420 REM reset system variable
430 LET address=65368: POKE 236
75,address-256*INT (address/256)
: POKE 23676,INT (address/256):
REM 16K address=32600
500 REM 1st character (original
)
510 DATA BIN 00111100
520 DATA BIN 01111110
530 DATA BIN 11011011
540 DATA BIN 01111110
550 DATA BIN 00111100
560 DATA BIN 00100100
570 DATA BIN 01000010
580 DATA BIN 10000001
600 REM 1st character (alternat
ive)
610 DATA BIN 00111100
620 DATA BIN 01111110
630 DATA BIN 11011011
640 DATA BIN 01111110
650 DATA BIN 00111100
660 DATA BIN 01000010
670 DATA BIN 00100100
680 DATA BIN 00011000
```

Program 4.12

Overprinting

Characters may be printed at PLOT co-ordinates using the CHARACTER PLOTTER routine. However, any character produced in this way is treated as an 8 × 8 block of pixels. So, if a bit is set to 0 in the character definition, the corresponding pixel on the screen is turned OFF. When the character is removed from the screen the whole 8 × 8 block is cleared so that all the corresponding pixels are turned OFF.

The OVERPRINTER routine treats the screen in a different way. OVERPRINTER preserves the states of the underlying pixels, and allows one character to be plotted on top of another, and then removed, without disturbing the underlying shape. Now any UDG character can move freely across the screen without disturbing the background, or interfering with other characters.

There are two ways to use OVERPRINTER: values of x and y, and the UDG character code are POKEd into 23296, 23297, and 23298 as for the CHARACTER PLOTTER routine, then

(a) POKE 1 into 23299 and plot the character; or
(b) POKE 0 into 23299 and erase the character, restoring the background at the same time.

The routine simply stores a copy of the background under the character as it is plotted, and then restores this when the character is erased. The background is not actually removed when the character is plotted, so any underlying background characters will 'show through' the top character if any of the top character's bits are set to 0.

For example, if a UDG is defined in the shape of a window, any background or any other characters over which the window is plotted will be visible 'through' the window once it has been drawn.

Moving a character across a background, or other character, requires the following steps:

(a) Set x and y, and the character code.
    Place 1 in 23299.
    Use OVERPRINTER to plot the character.
(b) Place 0 in 23299.
    Use OVERPRINTER to remove the character and restore the background.
(c) Calculate the next position for the character.

Repeat steps (a), (b), and (c) as often as is required. As each character is plotted, the underlying background is stored. When location 23299 is set to 0, and OVERPRINTER is used to erase a character, the stored background is copied into the position currently occupied by that character. Other tasks may be

programmed between printing a character and then erasing it, since the background is stored until it is required. Up to 21 different characters may be used, and a background will be stored for each. The backgrounds are stored and recalled in order of the character codes, so only one background may be stored at a time for any character.

The UDG system variable may be changed at any time, but since only 21 backgrounds may be stored, it is essential to keep track of the character sets. Switching character sets does not produce a separate store of backgrounds; nor does the routine make any distinction between the backgrounds under CHR$ 144 in the original UDG character set and CHR$ 144 in an alternate character set.

Several characters may be plotted on top of one another (up to 21 characters), as shown in Fig. 4.9, but it is important that these characters are removed in reverse order of printing, starting with the topmost character.

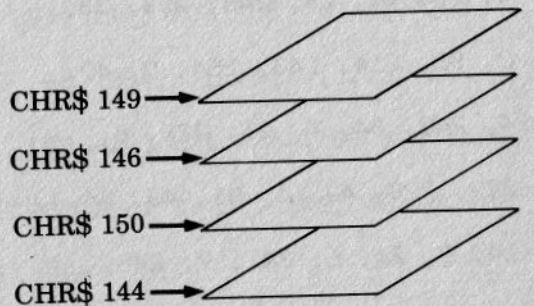
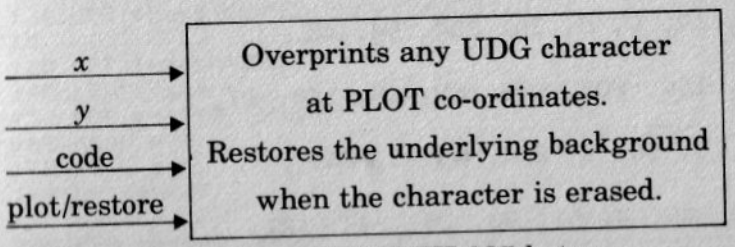


Figure 4.9 Characters plotted on top of one another

Remove these in order: CHR$ 149 then CHR$ 146, CHR$ 150, and CHR$ 144.

OVERPRINTER



LENGTH 285 bytes

Using the routine

POKE the *x* co-ordinate of the bottom left corner of the character into location 23296.
POKE the *y* co-ordinate of the bottom left corner of the character into location 23297.
POKE the character code into location 23298.
POKE 1 (to plot) or 0 (to erase) into location 23299.
1 plots the character and stores the background,
0 erases the character and restores the background.

Note: The routine uses 23300 and 23301 as workspace, and stores the backgrounds in locations 23382–23550.

Overprinter data

```
58, 2, 91, 254, 144, 56, 89, 254, 165, 48,

85, 42, 123, 92, 43, 214, 143, 17, 8, 0,

254, 0, 40, 4, 25, 61, 24, 248, 229, 33,

86, 91, 58, 2, 91, 214, 143, 254, 0, 40,

4, 25, 61, 24, 248, 34, 4, 91, 225, 6,

0, 126, 43, 229, 229, 42, 4, 91, 43, 34,

4, 91, 225, 14, 0, 24, 1, 241, 7, 245,

197, 245, 33, 0, 91, 126, 254, 249, 48, 89,

129, 79, 35, 126, 254, 8, 56, 81, 254, 169,

48, 77, 128, 71, 24, 6, 24, 73, 24, 223,

24, 205, 62, 16, 128, 71, 33, 0, 64, 254,

128, 48, 9, 17, 0, 8, 25, 254, 64, 48,

1, 25, 203, 184, 203, 176, 62, 63, 144, 17,

32, 0, 254, 8, 56, 5, 214, 8, 25, 24,

247, 254, 0, 40, 4, 61, 36, 24, 248, 121,
```



```
254, 8, 56, 5, 214, 8, 35, 24, 247, 79,

62, 7, 145, 24, 10, 24, 189, 24, 185, 24,

109, 24, 111, 55, 63, 87, 71, 94, 245, 229,

42, 4, 91, 126, 79, 225, 241, 254, 0, 40,

6, 203, 11, 203, 9, 16, 250, 58, 3, 91,

254, 0, 40, 39, 229, 42, 4, 91, 203, 193,

203, 67, 40, 2, 203, 129, 225, 241, 48, 2,

203, 195, 122, 66, 254, 0, 40, 6, 203, 3,

203, 1, 16, 250, 115, 229, 42, 4, 91, 113,

225, 24, 20, 203, 195, 203, 65, 40, 2, 203,

131, 241, 122, 66, 254, 0, 40, 4, 203, 3,

16, 252, 115, 193, 12, 121, 254, 8, 32, 153,

4, 241, 225, 120, 254, 8, 32, 143, 40, 4,

241, 193, 241, 225, 201
```

Example 4.13

```
 10 REM OVERPRINTING CHARACTERS
 20 REM set colours
 30 PAPER 7: INK 3: BORDER 7
 40 CLS
100 REM set new ramtop
110 CLEAR 65082: REM 16K=32314
120 LET start=65083: REM 16K=32
315
130 LET length=285
140 LET s=start
150 FOR i=1 TO length
160 READ n
170 POKE s,n
180 LET s=s+1
190 NEXT i
200 REM define UDGs
210 FOR i=1 TO 3
```

Program 4.13 *(continues)*

```
220 READ c$
230 FOR j=0 TO 7
240 READ n
250 POKE USR c$+j,n
260 NEXT j
270 NEXT i
300 REM place first UDG
310 POKE 23297,80: REM y
320 POKE 23298,144: REM code
330 POKE 23299,1: REM print
340 FOR x=100 TO 132 STEP 8
350 POKE 23296,x
360 RANDOMIZE USR start
370 NEXT x
400 REM move 2nd + 3rd UDGs
410 FOR j=0 TO 32
420 POKE 23296,152-2*j: REM x
430 POKE 23297,80: REM y
440 POKE 23298,145: REM code
450 POKE 23299,1: REM plot
460 RANDOMIZE USR start
470 PAUSE 10
480 POKE 23299,0: REM restore
490 RANDOMIZE USR start
500 POKE 23296,152-2*j-1: REM x
510 POKE 23298,146: REM code
520 POKE 23299,1: REM plot
530 RANDOMIZE USR start
540 PAUSE 10
550 POKE 23299,0: REM restore
560 RANDOMIZE USR start
570 NEXT j
600 REM end
610 PAPER 7: INK 0: BORDER 7
1000 REM OVERPRINTER
1010 DATA 58,2,91,254,144,56,89,
254,165,48
1020 DATA 85,42,123,92,43,214,14
3,17,8,0
1030 DATA 254,0,40,4,25,61,24,24
8,229,33
1040 DATA 86,91,58,2,91,214,143,
254,0,40
1050 DATA 4,25,61,24,248,34,4,91
,225,6
1060 DATA 0,126,43,229,229,42,4,
91,43,34
1070 DATA 4,91,225,14,0,24,1,241
,7,245
1080 DATA 197,245,33,0,91,126,25
4,249,48,89
1090 DATA 129,79,35,126,254,8,56
,81,254,169
1100 DATA 48,77,128,71,24,6,24,7
3,24,223
```



```
1110 DATA 24,205,62,16,128,71,33
,0,64,254
1120 DATA 128,48,9,17,0,8,25,254
,64,48
1130 DATA 1,25,203,184,203,176,6
2,63,144,17
1140 DATA 32,0,254,8,56,5,214,8,
25,24
1150 DATA 247,254,0,40,4,61,36,2
4,248,121
1160 DATA 254,8,56,5,214,8,35,24
,247,79
1170 DATA 62,7,145,24,10,24,189,
24,185,24
1180 DATA 109,24,111,55,63,87,71
,94,245,229
1190 DATA 42,4,91,126,79,225,241
,254,0,40
1200 DATA 6,203,11,203,9,16,250,
58,3,91
1210 DATA 254,0,40,39,229,42,4,9
1,203,193
1220 DATA 203,67,40,2,203,129,22
5,241,48,2
1230 DATA 203,195,122,66,254,0,4
0,6,203,3
1240 DATA 203,1,16,250,115,229,4
2,4,91,113
1250 DATA 225,24,20,203,195,203,
65,40,2,203
1260 DATA 131,241,122,66,254,0,4
0,4,203,3
1270 DATA 16,252,115,193,12,121,
254,8,32,153
1280 DATA 4,241,225,120,254,8,32
,143,40,4
1290 DATA 241,193,241,225,201
1300 REM UDG DEFINITIONS
1310 DATA "A",170,170,170,170,17
0,170,170,170
1320 DATA "B",60,126,219,126,60,
36,66,129
1330 DATA "C",60,126,219,126,60,
66,36,24
```

Program 4.13

5 DOMINOES

DOMINOES allows the Spectrum to play a really mean game of your grandmother's favourite pastime.

As presented, the game runs on a 48K Spectrum, but it will also run on a 16K machine with two very minor amendments. These amendments are detailed in the explanation which follows.

Because DOMINOES is such a large program, and because it has been written to fit into the 16K machine, the Basic code has been compacted wherever possible, using multiple statements on many program lines, and omitting REM statements. Unfortunately, this makes the program more difficult to read. However, the following notes offer an explanation of how the program works.

The program is constructed in three parts:

1. The main program, which loads the other two parts before the game begins.
2. A machine code routine, ROTATED SCALED CHARACTER PLOTTER, which is stored as CODE, using the name 'rotplot'. This routine is loaded by (1) when it starts.
3. The UDG character definitions which are stored as CODE, using the name 'udg', and which are loaded by (1) when it starts.

Code

The faces of the dominoes are plotted using the Rotated Scaled Character Plotter routine which allows each face to be accurately positioned, and to be rotated before plotting, if necessary. Each of the domino faces shown in Fig. 5.1 is stored as a UDG character; A, B, C, D, E, F, or G.

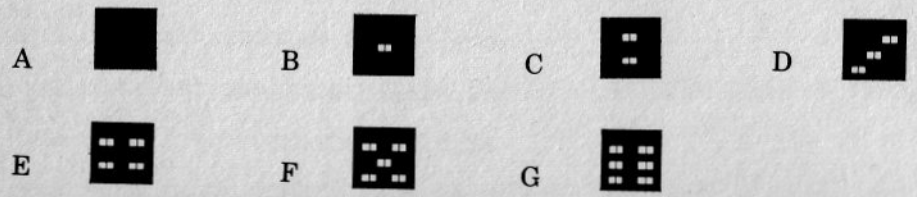


Figure 5.1 The domino faces

When the game is being played, each domino will consist of two separate UDG characters, one for each end face, as shown in Fig. 5.2.

Figure 5.2 A complete domino

A single program (5.1) is used to place the CODE for the plotting routine and the UDG definitions in memory:

```
 10 REM ROUTINES FOR DOMINOES
100 REM SET NEW RAMTOP
110 CLEAR 32185
120 LET start=32186
130 LET length=414
140 LET s=start
150 FOR i=1 TO length
160 READ n
170 POKE s,n
180 LET s=s+1
190 NEXT i
200 REM DEFINE UDG CHARACTERS
210 FOR i=1 TO 7
220 READ c$
230 FOR j=0 TO 7
240 READ d
250 POKE USR c$+j,d
260 NEXT j
270 NEXT i
5000 REM ROTATED SCALED CHARACTE
R PLOTTER
5010 DATA 58,5,91,254,4,48,43,58
,2,91
5020 DATA 254,32,56,36,254,128,5
6,8,254,144
5030 DATA 56,28,254,165,56,7,33,
7,61,214
5040 DATA 32,24,6,42,123,92,43,2
14,143,17
5050 DATA 8,0,254,0,40,6,25,61,2
4,248
5060 DATA 24,104,58,5,91,254,0,3
2,10,17
5070 DATA 17,91,1,8,0,237,184,24
,82,254
5080 DATA 2,32,25,6,8,17,10,91,1
97,26
5090 DATA 79,126,6,8,23,203,25,1
6,251,121
```

Program 5.1 *(continues)*

```
5100 DATA 18,193,19,43,16,238,24
,53,254,3
5110 DATA 32,25,6,8,17,17,91,126
,197,6
5120 DATA 8,23,235,78,203,25,113
,235,27,16
5130 DATA 246,193,43,16,235,24,2
4,6,8,17
5140 DATA 17,91,126,197,6,8,203,
31,235,78
5150 DATA 203,17,113,235,27,16,2
45,193,43,16
5160 DATA 234,33,17,91,229,24,2,
24,70,33
5170 DATA 0,91,126,79,35,126,71,
33,4,91
5180 DATA 126,35,35,35,119,35,35
,62,0,119
5190 DATA 225,24,23,241,193,225,
43,229,33,4
5200 DATA 91,126,35,35,35,119,22
5,24,3,241
5210 DATA 193,225,58,0,91,79,126
,229,197,245
5220 DATA 33,8,91,62,0,119,241,7
,245,33
5230 DATA 3,91,126,35,35,35,119,
24,8,24
5240 DATA 103,24,206,24,220,24,2
35,62,16,128
5250 DATA 71,33,0,64,254,128,48,
9,17,0
5260 DATA 8,25,254,64,48,1,25,20
3,184,203
5270 DATA 176,62,63,144,17,32,0,
254,8,56
5280 DATA 5,214,8,25,24,247,254,
0,40,4
5290 DATA 61,36,24,248,121,254,8
,56,5,214
5300 DATA 8,35,24,247,79,62,7,14
5,55,63
5310 DATA 87,71,94,254,0,40,4,20
3,11,16
5320 DATA 252,241,245,56,4,203,1
31,24,2,203
5330 DATA 195,122,66,254,0,40,4,
203,3,16
5340 DATA 252,115,24,8,24,77,24,
149,24,149
5350 DATA 24,149,33,6,91,126,61,
119,254,0
5360 DATA 40,7,241,193,12,197,24
5,24,134,33
5370 DATA 8,91,126,60,119,254,8,
```



```
40,7,241
5380 DATA 193,12,197,245,24,220,
33,7,91,126
5390 DATA 61,119,254,0,40,7,241,
193,4,197
5400 DATA 245,24,201,33,9,91,126
,60,119,254
5410 DATA 8,40,7,241,193,4,197,2
45,24,182
5420 DATA 241,193,225,201
5500 REM UDG DEFINITIONS
5510 DATA "A",255,255,255,255,25
5,255,255,255
5520 DATA "B",255,255,255,255,23
1,255,255,255
5530 DATA "C",255,255,255,231,25
5,231,255,255
5540 DATA "D",255,255,249,255,23
1,255,159,255
5550 DATA "E",255,255,153,255,25
5,255,153,255
5560 DATA "F",255,255,153,255,23
1,255,153,255
5570 DATA "G",255,255,153,255,15
3,255,153,255
```

Program 5.1

Running the program creates the CODE for the plotting routine and for the UDG definitions. These two blocks of CODE must be saved separately. Note that line 120 of the program sets RAMTOP to 32186 which allows the program to be used in 48K or 16K machines without making any changes.

The plotting routine should be saved using the SAVE command for 48K and 16K machines:

SAVE "rotplot" CODE 32186,414

and this should be saved as the *second* program on the tape. The UDG definitions will occupy different positions in memory in the 48K and 16K machines, and should be saved as the *third* program on the tape, using

(48K) (16K)

SAVE "udg" CODE 65368,168 SAVE "udg" CODE 32600,168

The program used to create the two blocks of CODE is no longer required, but common sense suggests that it should be saved on a blank cassette and filed carefully away just in case it is needed at some future date!

Main program

Enter Program 5.2, and save it as the first program on the tape. 16K owners should make one change, to line 130:

```
130 INK 7: PRINT AT 0,0: LOAD "
rotplot"CODE : PRINT AT 0,0: LOAD
D "udg"CODE 32600,168: INK 0
```

The Spectrum keeps track of each domino using an array, $d(i,j)$, in which a 1 denotes a domino which is still available for play, and a 0 denotes a domino which has already been used. There are 28 dominoes available at the start of the game, and these are shown in the table below which represents the array $d(i,j)$. Note that the element numbers are always 1 more than the values of the faces of the dominoes, simply because the Spectrum does not allow a zero subscript in an array (i.e., $d(0,0)$ is not allowed). So, instead of $d(0,0)$ the element $d(1,1)$ must be used to represent the domino which has 0 spots on each face, which in turn demands that the array subscripts be one more than the number of spots throughout the array.

In the table, the first number in each pair represents the number of spots on the left-hand face of the domino, while the second number represents the number on the right.

		second subscript (j)						
		1	2	3	4	5	6	7
	1	0,0	0,1	0,2	0,3	0,4	0,5	0,6
	2		1,1	1,2	1,3	1,4	1,5	1,6
first	3			2,2	2,3	2,4	2,5	2,6
subscript	4				3,3	3,4	3,5	3,6
(i)	5					4,4	4,5	4,6
	6						5,5	5,6
	7							6,6

When the game begins, the contents of the array are set to:

		second subscript (j)						
		1	2	3	4	5	6	7
	1	1	1	1	1	1	1	1
	2	0	1	1	1	1	1	1
first	3	0	0	1	1	1	1	1
subscript	4	0	0	0	1	1	1	1
(i)	5	0	0	0	0	1	1	1
	6	0	0	0	0	0	1	1
	7	0	0	0	0	0	0	1

Players' hands are dealt randomly from the available dominoes, at the start of the game, and a 0 is placed in the array whenever a domino is chosen.

Before play begins, the computer searches the player's hand and the computer's own hand, to find the 'highest' domino. 'Highest' is defined as

(a) the double with the greatest number of spots, i.e., double 6, then double 5, . . .
(b) if there are no doubles, choose the domino with the greatest total number of spots on its faces.
(c) if the 'highest' dominoes in each hand are equal in value, a random choice is made to decide who plays first.

The player with the 'highest' domino must play first. Players then place dominoes alternately. Dominoes may be placed at either end of the line of dominoes in the playing area, by specifying which domino is to be played from the player's hand (1–6). The computer identifies the end at which that domino should be placed, and then plots it there. In any case in which the domino could be placed at either end, it is placed at the left-hand end.

When a player has had the opportunity of playing, the computer picks a domino at random from those remaining, and adds it to the hand, if necessary.

When one player has played all the dominoes in the hand, and there are no remaining dominoes which can be added to that hand, play stops and that player is declared the winner.

If both players 'knock', one after the other, play cannot continue, so the score is calculated for each player's hand, and this is used to

decide the winner. Each player's score is calculated by adding the number of spots on the remaining dominoes in the hand. The winner is the player with the lowest score.

```
  10 REM DOMINOES
  20 PAPER 7: INK 0: BORDER 7: B
RIGHT 1
 100 CLEAR 32185
 110 LET start=32186
 120 PRINT AT 10,12;"DOMINOES"
 130 INK 7: PRINT AT 0,0: LOAD "
rotplot"CODE : PRINT AT 0,0: LOA
D "udg"CODE 65368,168: INK 0
 200 DIM d(7,7): DIM r(2,6): DIM
 l(2,6): DIM n(2): DIM k(2): DIM
 t(2): DIM w(2)
 210 LET e$="
             "
 300 LET b0=23296: LET b1=23297:
 LET b2=23298: LET b3=23299: LET
 b4=23300: LET b5=23301
 310 CLS : GO SUB 600: GO SUB 70
0: GO SUB 900: GO SUB 1000: GO S
UB 1200: GO SUB 1400: GO SUB 170
0
 400 REM MAIN CONTROL LOOP
 410 LET p=3-p: LET k(p)=0
 420 IF p=1 THEN  GO SUB 2000
 430 IF p=2 THEN  GO SUB 2500
 440 IF d=7 THEN  LET k(p)=1: GO
 SUB 2300
 450 IF d<>7 THEN  GO SUB 2600
 460 IF d<>7 THEN  GO SUB 3300
 470 GO SUB 1000: GO SUB 1200
 500 IF k(1)=1 AND k(2)=1 THEN
GO TO 3500
 510 IF n(p)=0 THEN  GO TO 3500
 520 GO TO 410
 600 REM GENERATE DOMINOES
 610 FOR i=1 TO 7: FOR j=i TO 7:
 LET d(i,j)=1: NEXT j: NEXT i: L
ET t=28: RETURN
 700 REM ISSUE HANDS
 710 RANDOMIZE 0: FOR p=1 TO 2:
FOR i=1 TO 6: GO SUB 800: LET r(
p,i)=n1: LET l(p,i)=n2: NEXT i:
NEXT p: LET n(1)=6: LET n(2)=6:
RETURN
 800 REM GENERATE SPOTS
 810 IF t=0 THEN  GO TO 850
 820 LET n1=INT (RND*7): LET n2=
INT (RND*7)
 830 IF d(n1+1,n2+1)=0 THEN  GO
TO 820
```

Program 5.2 *(continues)*


```
 840 LET d(n1+1,n2+1)=0: LET t=t
-1
 850 RETURN
 900 REM DRAW FIELD OF PLAY
 910 PLOT 255,151: DRAW 0,-119:
DRAW -255,0: DRAW 0,119: DRAW 25
5,0: RETURN
1000 REM DRAW COMPUTER'S HAND
1010 PRINT AT 1,0;e$: POKE b1,16
0: POKE b2,144: POKE b3,1: POKE
b4,1: IF n(2)=0 THEN  GO TO 1030
1020 FOR i=1 TO n(2): PRINT AT 1
,4*(i-1);i: POKE b0,8+32*(i-1):
POKE b5,3: RANDOMIZE USR start:
POKE b0,16+32*(i-1): POKE b5,1:
RANDOMIZE USR start: NEXT i
1030 RETURN
1200 REM DRAW PLAYER'S HAND
1210 PRINT AT 19,0;e$+e$: POKE b
1,16: POKE b3,1: POKE b4,1: IF n
(1)=0 THEN  GO TO 1230
1220 FOR i=1 TO n(1): PRINT AT 1
9,4*(i-1);i: POKE b0,8+32*(i-1):
 POKE b2,l(1,i)+144: POKE b5,3:
RANDOMIZE USR start: POKE b0,16+
32*(i-1): POKE b2,r(1,i)+144: PO
KE b5,1: RANDOMIZE USR start: NE
XT i
1230 RETURN
1400 REM DECIDE WHO PLAYS FIRST
1410 REM SEARCH FOR DOUBLES
1420 LET p=0: FOR i=6 TO 0 STEP
-1: FOR j=1 TO 6
1430 IF r(1,j)=i AND l(1,j)=i TH
EN  LET p=1: LET d=j: LET j=6: L
ET i=0
1440 IF r(2,j)=i AND l(2,j)=i TH
EN  LET p=2: LET d=j: LET j=6: L
ET i=0
1450 NEXT j: NEXT i
1460 IF p<>0 THEN  GO TO 1580
1500 REM FIND HIGHEST DOMINO
1510 LET t(1)=0: LET t(2)=0: LET
 w(1)=0: LET w(2)=0
1520 FOR i=1 TO 2: LET t1=0: LET
 d=0: FOR j=1 TO 6: LET t1=r(i,j
)+l(i,j)
1530 IF t1>t(i) THEN  LET t(i)=t
1: LET w(i)=j
1540 NEXT j: NEXT i
1550 IF t(1)>t(2) THEN  LET p=1:
 LET d=w(1)
1560 IF t(1)<t(2) THEN  LET p=2:
 LET d=w(2)
1570 IF t(1)=t(2) THEN  LET p=1+
```

```
INT (RND*2): LET d=w(p)
1580 GO SUB 2400: RETURN
1700 REM PLOT FIRST DOMINO
1710 BEEP .15,24
1720 IF p=2 THEN  LET p$="I WILL
"
1730 IF p<>2 THEN  LET p$="YOU M
UST"
1740 PRINT AT 20,0;p$;" PLAY NO.
 ";d;" FIRST."
1750 POKE b0,120: POKE b1,40: PO
KE b2,l(p,d)+144: POKE b3,1: POK
E b4,1: POKE b5,3: RANDOMIZE USR
 start: POKE b0,128: POKE b2,r(p
,d)+144: POKE b5,1: RANDOMIZE US
R start: LET rx=129: LET ry=40:
LET rv=r(p,d): LET lx=119: LET l
y=40: LET lv=l(p,d): GO SUB 3300
: GO SUB 2400: RETURN
2000 REM ASK PLAYER WHICH DOMINO
2010 BEEP .15,24: PRINT AT 20,0;
e$;AT 20,0;"WHICH DOMINO?(1-";n(
1);" or k TO KNOCK)"
2020 LET k$=INKEY$: IF k$="" THE
N  GO TO 2020
2030 PRINT AT 20,0;e$
2040 IF k$="k" THEN  LET d=7: GO
 TO 2080
2050 IF CODE (k$)<49 OR CODE (k$
)>48+n(1) THEN  GO TO 2010
2060 LET d=CODE (k$)-48
2070 IF r(1,d)<>lv AND r(1,d)<>r
v AND l(1,d)<>lv AND l(1,d)<>rv
THEN  GO SUB 2200: GO TO 2010
2080 RETURN
2200 REM INVALID CHOICE
2210 BEEP .15,24: PRINT AT 20,0;
e$;AT 20,0;"YOU CAN'T CHOOSE THA
T DOMINO.": GO SUB 2400: BEEP .1
5,24: PRINT AT 20,0;e$;AT 20,0;"
TRY AGAIN....": GO SUB 2400: RET
URN
2300 REM KNOCK
2310 BEEP .15,12: PAUSE 10: BEEP
 .15,12: PRINT AT 20,0;e$;AT 20,
0;"KNOCK...KNOCK.....": PAUSE 10
: BEEP .15,12: PAUSE 10: BEEP .1
5,12: RETURN
2400 REM PAUSE
2410 FOR w=1 TO 250: NEXT w: RET
URN
2500 REM SELECT COMPUTER'S DOMIN
O
2510 LET d=7
2520 FOR i=1 TO n(2)
```

Program 5.2 *(continues)*


```
2530 IF r(2,i)=rv OR r(2,i)=lv O
R l(2,i)=rv OR l(2,i)=lv THEN  L
ET d=i: LET i=n(2)
2540 NEXT i
2550 RETURN
2600 REM PLOT DOMINO
2610 IF r(p,d)=rv THEN  LET e=1:
 LET f=r(p,d): LET s=l(p,d)
2620 IF l(p,d)=rv THEN  LET e=1:
 LET f=l(p,d): LET s=r(p,d)
2630 IF r(p,d)=lv THEN  LET e=2:
 LET f=r(p,d): LET s=l(p,d)
2640 IF l(p,d)=lv THEN  LET e=2:
 LET f=l(p,d): LET s=r(p,d)
2650 IF e=1 THEN  GO SUB 2900
2660 IF e=2 THEN  GO SUB 3100
2670 POKE b0,ex+dx: POKE b1,ey+d
y: POKE b2,f+144: POKE b3,1: POK
E b4,1: POKE b5,rot2: RANDOMIZE
USR start: POKE b0,ex+dx+dx: POK
E b1,ey+dy+dy: POKE b2,s+144: PO
KE b5,rot1: RANDOMIZE USR start:
 RETURN
2900 REM PLOT AT RIGHT END
2910 LET ex=rx: LET ey=ry: LET d
x=8: LET dy=0: LET rot1=1: LET r
ot2=3
2920 IF ex=231 AND ey=40 THEN  L
ET ex=230: LET ry=41: LET ey=41:
 LET dx=0: LET dy=8: LET rot1=0:
 LET rot2=2: GO TO 2940
2930 IF ex>228 AND ey<>40 THEN
LET ex=230: LET dx=0: LET dy=8:
LET rot1=0: LET rot2=2
2940 IF ey=143 AND ex=230 THEN
LET ey=142: LET ry=142: LET rx=2
29: LET ex=229: LET dx=-8: LET d
y=0: LET rot1=1: LET rot2=3: GO
TO 2960
2950 IF ey=142 THEN  LET dx=-8:
LET dy=0: LET rot1=1: LET rot2=3
2960 LET rx=rx+dx+dx+INT (dx/8):
 LET ry=ry+dy+dy+INT (dy/8): LET
 rv=s: RETURN
3100 REM PLOT AT LEFT END
3110 LET ex=lx: LET ey=ly: LET d
x=-8: LET dy=0: LET rot1=1: LET
rot2=3
3120 IF ex=17 AND ey=40 THEN  LE
T ex=18: LET ly=41: LET ey=41: L
ET dx=0: LET dy=8: LET rot1=0: L
ET rot2=2: GO TO 3140
3130 IF ex<33 AND ey<>40 THEN  L
ET ex=18: LET dx=0: LET dy=8: LE
T rot1=0: LET rot2=2
```

```
3140 IF ey=143 AND ex=18 THEN  L
ET ey=142: LET ly=142: LET lx=19
: LET ex=19: LET dx=8: LET dy=0:
 LET rot1=3: LET rot2=1: GO TO 3
160
3150 IF ey=142 THEN  LET dx=8: L
ET dy=0: LET rot1=3: LET rot2=1
3160 LET lx=lx+dx+dx+INT (dx/8):
 LET ly=ly+dy+dy+INT (dy/8): LET
 lv=s: RETURN
3300 REM MOVE UP, AND FETCH NEW
DOMINO
3310 IF d=n(p) THEN  GO TO 3330
3320 FOR i=d TO n(p)-1: LET r(p,
i)=r(p,i+1): LET l(p,i)=l(p,i+1)
: NEXT i
3330 IF t=0 THEN  LET n(p)=n(p)-
1: GO TO 3350
3340 GO SUB 800: LET r(p,n(p))=n
1: LET l(p,n(p))=n2
3350 RETURN
3500 REM END OF GAME
3510 IF n(1)=0 THEN  GO SUB 4100
3520 IF n(2)=0 THEN  GO SUB 4100
3530 IF k(1)=1 AND k(2)=1 THEN
LET k(1)=0: LET k(2)=0: GO SUB 4
000: GO SUB 2400: GO SUB 4100
3540 GO SUB 2400: PRINT AT 20,0;
e$;AT 20,0;"ANOTHER GAME? (y or
n)"
3550 LET k$=INKEY$: IF k$="" THE
N  GO TO 3550
3560 IF k$="y" THEN  GO TO 300
3570 IF k$<>"y" THEN  GO TO 7000
3580 STOP
4000 REM BOTH KNOCKING
4010 BEEP .15,24: PRINT AT 20,0;
e$;AT 20,0;"NEITHER OF US CAN PL
AY": GO SUB 2400: PRINT AT 20,0;
e$;AT 20,0;"SO I WILL ADD THE DO
TS WE HOLD": GO SUB 4200: RETURN

4100 REM RESULT
4110 IF p=2 THEN  LET w$="YIPPEE
' I AM THE WINNER!"
4120 IF p=1 THEN  LET w$="OOPS!
YOU ARE THE WINNER."
4130 IF p=0 THEN  LET w$="GOSH!
IT'S A DRAW!"
4140 PRINT AT 20,0;e$;AT 20,0;w$
: RETURN
4200 REM ADD REMAINING DOTS
4210 FOR p=1 TO 2: LET t(p)=0: F
OR i=1 TO n(p): LET t(p)=t(p)+r(
p,i)+l(p,i): NEXT i: NEXT p
```



```
4220 IF t(1)>t(2) THEN  LET p=2
4230 IF t(1)<t(2) THEN  LET p=1
4240 IF t(1)=t(2) THEN  LET p=0
4250 RETURN
7000 REM END OF PROGRAM
```

Program 5.2

Program notes

10–20		Set colours
100–130		Print a title, then load the CODE for rotplot and the UDG definitions. Setting INK 7 ensures that messages are printed on the screen invisibly.
200–210		Dimension arrays, then define a string of blanks which can be used to erase a line on the screen.
300–310		Initialize
	300	Locations 23296–23301 will be used to pass values to the routine used to rotate and plot the faces of the dominoes. To save program space, and to speed execution, define variables *b*0 to *b*5 to represent these locations.
	310	Generate dominoes; issue hands; draw field of play; draw computer's hand; draw player's hand; decide who plays first; plot first domino.
400–520		Main control loop.
	410	p = 1 or 0, alternately. 1 represents the human player, 0 represents the computer.
	420	Fetch player's choice of domino (1–6, or 7 if knocking), if it is the player's turn.
	430	Fetch the computer's choice of domino, if it is the computer's turn.
	440	d is the domino number. $k(1)$ and $k(2)$ are set to 1 if the player or computer chooses to knock. If d = 7 then set the knock flag, and knock.
	450	If not knocking, plot domino d.
	460	If not knocking, shift dominoes in the hand along, and replace any which have been played.
	470	Draw computer's hand; draw player's hand.

500 If the computer and the player are both knocking, exit via scoring routine.

510 If there are no dominoes left in the current hand, exit via the scoring routine.

520 Go to the start of the loop.

600–610 Generate dominoes by placing ones in the appropriate positions in array $d(i,j)$, to indicate these values are available for play. Total number of dominoes, t, is 28.

700–710 Randomly choose the dominoes to be transferred from $d(i,j)$ to each hand. Dominoes are chosen by the subroutine at 800, which returns two whole numbers $n1$ and $n2$, both in the range 0–6 inclusive. Hands consist of two arrays, $r(p,i)$ and $l(p,i)$.
$p = 1$ when dealing with the player's hand, and
$p = 0$ when dealing with the computer's hand.
i is the domino number (0–6) as shown on the screen.
$n(1)$ and $n(2)$ denote the number of dominoes left in each hand (6, initially).

800–850 Randomly generate the number of spots on each face of a domino. $n1$ and $n2$ represent the number of spots on the right and left ends respectively. Each chosen pair of values is checked against the appropriate entry in $d(i,j)$, to ensure that a valid number of spots has been generated.

900–910 Draw the field of play.

1000–1030 Draw the computer's hand using the ROTATED SCALED CHARACTER ROUTINE. The right-hand ends of the dominoes are plotted with rotation = 1, and the left-hand ends with rotation = 3. So that the human player cannot gain an advantage by seeing the computer's hand, blank characters are plotted, to represent the rear of the dominoes.

1200–1230 Draw the human player's hand. Use the same technique as for the computer's hand, but show the faces of the dominoes.

1400–1580 Decide which player must play first, and which domino must be played.

1420–1460 Search the hands for doubles, in descending order of the total number of spots. i.e., 6,6, 5,5, 4,4, etc. If the highest double is identified, return with p = player number, and d = domino number. If no double is found in either hand, move on to the next section.

1500–1580 Add the spots on each domino in both hands, and try to identify the 'highest' domino (i.e., the domino with the greatest number of spots). If the highest total in both hands is equal, choose which player plays first, at random. p = player number, and d = domino number.
Pause, then return.

1700–1750 Plot the first domino, with a suitable message.
$l(p,d)$ and $r(p,d)$ hold the number of spots on the right and left-hand ends of the domino.
The left face is plotted at 120,40 with rotation 3, while the right face is plotted at 128,40 and has rotation 1. After plotting the domino, the following variables are defined:
rx = x co-ordinate of the next right-hand plotting position.
ry = y co-ordinate of the next right-hand plotting position.
rv = number of spots on the last-plotted right-hand face.
lx = x co-ordinate of the next left-hand plotting position.
ly = y co-ordinate of the next left-hand plotting position.
lv = number of spots on the last-plotted left-hand face.
Use the subroutine at 3300 to replace the plotted domino in the hand from which it was taken. Pause, using the subroutine at 2400, before returning.

2000–2080 Ask the player which domino is to be played.

2040–2060 Check the response and reject it if it is not '*k*', or if it exceeds the number of dominoes left in the hand.

2070 Check that the domino chosen matches the left-hand or right-hand face of the line of dominoes already played.
On return from this subroutine, $d = 7$ if the player is knocking; otherwise d = domino number.

2200–2210 Print a message to indicate that an invalid choice of domino has been made, then invite the player to try again.

2300–2310 Print a message, and beep, to announce a 'knock'.

2400–2410 Pause.

2500–2520 Select which domino the computer will play.

2510 Begin by assuming the computer will knock ($d = 7$). Search the faces of the dominoes in the computer's hand, trying to match them against the rightmost and leftmost faces of the dominoes which lie at the right and left ends of the dominoes in play. If a match is found, set d to the number corresponding to the matching domino.

2600–2670 Plot domino d from the hand of player p.

2610–2640 Identify the end at which plotting will take place, setting e accordingly (1 = right, 2 = left).

2650–2660 Set parameters required to plot the domino at the appropriate end.

2670 Plot the domino.

2900–2960 Set parameters required for plotting at the right-hand end.

2910 Begin with $ex = rx$ x co-ordinate at which to begin plotting.
$ey = ry$ y co-ordinate at which to begin plotting
$dx = 8$ x increment between faces.
$dy = 0$ y increment between faces.
rot1 = 1 orientation of right face.
rot2 = 3 orientation of left face

2920 If the domino will fall partly outside the right edge of the playing area, change to a vertical domino, heading up the screen.

2930 If the domino is to be plotted vertically, set parameters accordingly.

2940 If the domino is to be plotted vertically, but will fall partly outside the top edge of the playing area, change to a horizontal domino, heading to the left.

2950 If the domino is to be plotted horizontally along the top of the playing area, set parameters accordingly.

2960 Calculate the right-hand x and y co-ordinates, for the next domino, and set rv to the number of spots on the right-hand face of the last domino plotted at the right end.

3100–3160 Set parameters required for plotting at the left-hand end.

3110 Begin with $ex = lx$ x co-ordinate at which to begin plotting.
$ey = ly$ y co-ordinate at which to begin plotting
$dx = -8$ x increment between faces.
$dy = 0$ y increment between faces.
rot1 = 1 orientation of right face.
rot2 = 3 orientation of left face

3120 If the domino will fall partly outside the left edge of the playing area, change to a vertical domino heading up the screen.

3130 If the domino is to be plotted vertically, set parameters accordingly.

3140 If the domino is to be plotted vertically, but will fall partly outside the top edge of the playing area, change to a horizontal domino heading to the right.

3150	If the domino is to be plotted horizontally along the top of the playing area, set parameters accordingly.
3160	Calculate the left-hand x and y co-ordinates for the next domino, and set lv to the number of spots on the leftmost face of the domino last plotted on the left side.
3300–3350	Adjust hand after playing a domino.
3300–3330	When a domino has been removed from a hand, fill the space with the next domino in the hand. Repeat, so that the following dominoes move along one position in the hand.
3340	Use the subroutine at 800 to generate a replacement domino.
3500–3580	End of game.
3510–3520	Check if a player has played all the dominoes in the hand.
3530	Check for both players knocking.
3540–3570	Another game? Begin again, or end the program.
4000–4010	Both knocking. Print message, then use the subroutine at 4200 to identify the player whose dominoes have the smallest total number of spots.
4100–4140	Announce the result.
4200–4250	Using $t(1)$ and $t(2)$ to hold the total number of spots in each hand, add the spots in each hand separately. Set p, to indicate the winning player (1 or 2). $p = 0$ indicates a draw.
7000	End.

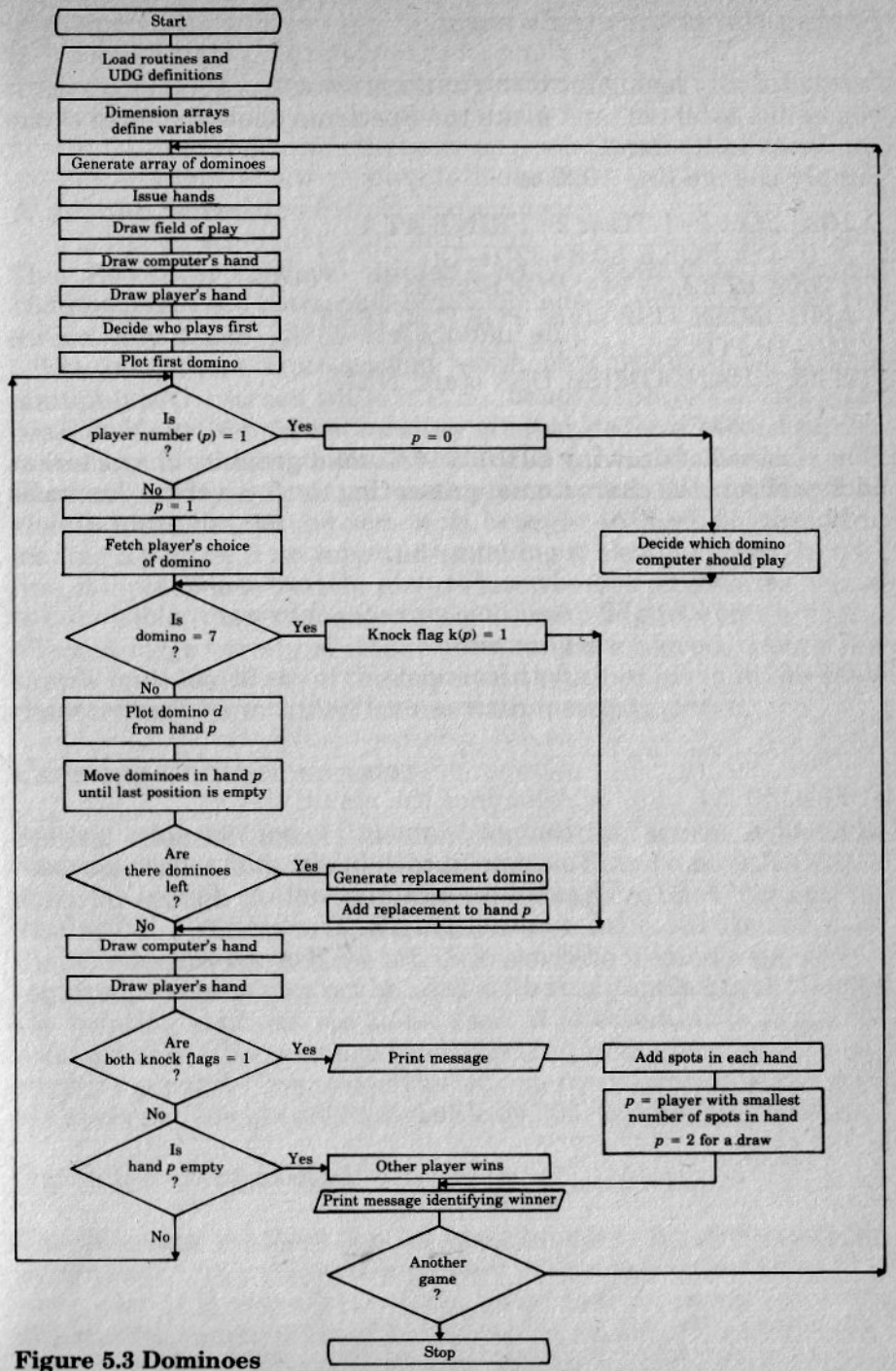


Figure 5.3 Dominoes

Seeing the computer's hand

Instead of only seeing the blank backs of the computer's dominoes, it is possible to 'cheat' and make the Spectrum show the faces of the dominoes in its hand.
Simply change line 1020 to

```
1020 FOR i=1 TO n(2): PRINT AT 1
,4*(i-1);i: POKE b0,8+32*(i-1):
POKE b2,l(2,i)+144: POKE b5,3: R
ANDOMISE USR start: POKE b0,16+3
2*(i-1): POKE b2,r(2,i)+144: POK
E b5,1: RANDOMISE USR start: NEX
T i
```

Now, instead of drawing CHR$ 144 (a solid graphics character) at each position, the characters representing the faces of the dominoes are used.

6 MULTIPLE SCREENS

A second screen

The Spectrum always displays the contents of locations 16384–22527 (the bytes containing the pixels which make up the screen) and 22528–23295 (the attributes).

However, there are occasions when it is useful to be able to manipulate the screen while it is not being displayed. This is only possible if another area of memory is set aside for a 'second screen' consisting of a second display file and a second attribute file. Characters may be printed on this screen, but these will not be visible until the second screen is brought into view. Instead of loading a picture from tape and watching it slowly appear, line by line, it is possible to load the picture into a second screen area, which is not visible, then suddenly reveal it, resulting in a much neater effect. A large variety of other effects can be achieved, using a few simple routines. Many of the important routines given in the other chapters may be modified for use with a second screen.

Creating another screen

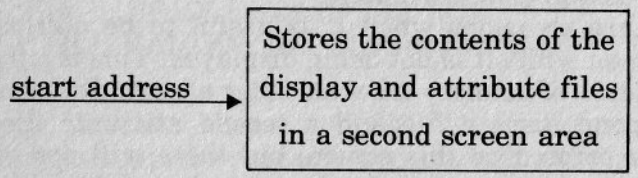
Setting aside an area of memory for another screen is just like reserving space for a machine code routine. Simply move RAMTOP down far enough to allow sufficient space to store 6144 bytes for the display file + 768 bytes for the attributes (total = 6912 bytes). Note that the display file and the attributes may be treated separately if required, and both may not be needed for every application. If any of the printing routines are to be used, it is essential to begin the second display file at a page boundary (i.e., in the first location of a page of memory). This means that the memory location at the start of the display file should be divisible by 256, leaving no remainder.

Storing and recalling

The following routines can be used to store the display and/or attribute files in a second screen area, or to recall information from store. Storing involves copying the contents of the normally-visible display and attribute files into the second screen area. This is done destructively, so that any previous contents in the second screen

area are erased as this area is filled with a copy of the display file. The display file remains intact. When storing has taken place in this way, the display file and the second screen area both have the same contents. The recall process works in the same way, destroying the contents of the display file as it copies the contents of the second screen area into the display file. When the recall routine has finished, both the display file and the second screen area have the same contents.

STORE DISPLAY AND ATTRIBUTES



LENGTH 13 bytes

Using the routine

POKE the start address of the second screen into locations 23296 and 23297.

Store display and attributes data

237, 91, 0, 91, 33, 0, 64, 1, 0, 27,
237, 176, 201

If START represents the start address of the second display file; IF START /256 =INT(START/256) THEN START is a page boundary. Repeatedly adding 256 to 16384 provides the page boundaries. Appendix A lists page boundaries, for convenience.

(48K)

RAMTOP = 65367 normally, so this should be moved to
65367 − 6912 = 58455
However, this would not allow the second display area to begin at a page boundary (i.e., the next location is 58456, which is not a page boundary). The next, lower, page boundary is at 58368, so set RAMTOP to 58368−1 = 58367.
The start of the second display file is then at 58368.

(16K)

RAMTOP = 32599 normally, so this should be moved to
32599 − 6912 = 25687
However, this would not allow the second display area to begin at a page boundary (i.e., the next location is 25689, which is not a

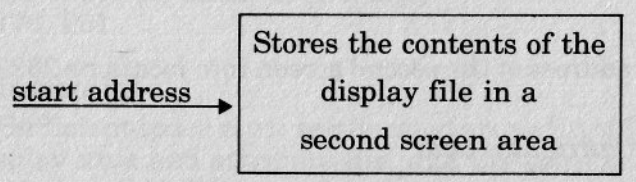
page boundary). The next, lower, page boundary is at 25600, so set RAMTOP to 25600 − 1 = 25599.
The start of the second display file is then at 25600.

If a second attribute file is to be created, without a second display file, page boundaries do not need to be taken into account. Simply move RAMTOP down by 768 bytes.

The routines which follow demand that the address of the start of the second display file or attribute file is placed in locations 23296 and 23297. The low byte should go in 23296, and the high byte in 23297. If screen2 denotes the address of the start of the display or attribute file, this can be done in BASIC, using

POKE 23296, screen2 − 256*INT(screen2/256)
POKE 23297, INT(screen2/256)

STORE DISPLAY



LENGTH 13 bytes

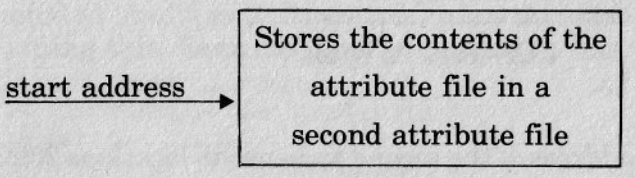
Using the routine

POKE the start address of the second screen into locations 23296 and 23297.

Store display data

237, 91, 0, 91, 33, 0, 64, 1, 0, 24,
237, 176, 201

STORE ATTRIBUTES



LENGTH 13 bytes

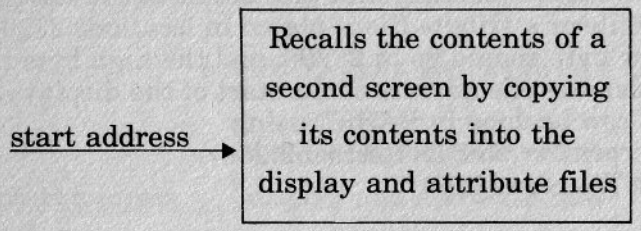
Using the routine

POKE the start address of the second attribute file into locations 23296 and 23297.

Store attributes data

237, 91, 0, 91, 33, 0, 88, 1, 0, 3,
237, 176, 201

RECALL DISPLAY AND ATTRIBUTES



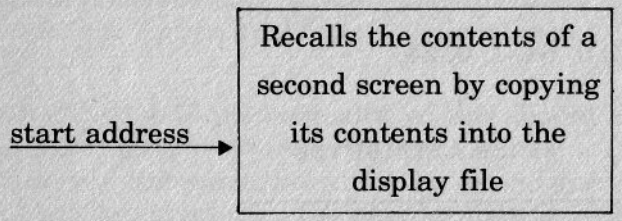
LENGTH 12 bytes

Using the routine
POKE the start address of the second screen into locations 23296 and 23297.

Recall display and attributes data

42, 0, 91, 17, 0, 64, 1, 0, 27, 237,
176, 201

RECALL DISPLAY



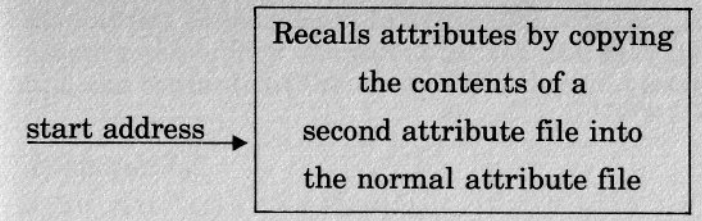
LENGTH 12 bytes

Using the routine
POKE the start address of the second screen into locations 23296 and 23297.

Recall display data

42, 0, 91, 17, 0, 64, 1, 0, 24, 237,
176, 201

RECALL ATTRIBUTES



LENGTH 12 bytes

Using the routine
POKE the start address of the second attribute file into locations 23296 and 23297.

Recall attributes data

42, 0, 91, 17, 0, 88, 1, 0, 3, 237,
176, 201

Example 6.1

Sufficient space must be allocated above RAMTOP for the second display area and attribute file, and the store and recall routines. Lowering RAMTOP to the nearest convenient page boundary leaves sufficient space for the routines in this case, but if there was insufficient room for the routines, RAMTOP would have to be lowered to the next lower page boundary – or to an even lower page boundary if the routines required even more space. In this particular example, nothing is being printed on the second screen (it is merely being stored and recalled) so it is not essential that page boundaries are taken into account, but it is good practice, and allows the routines to fit in quite neatly as well.

Since two routines are being used, the start address of each routine must be set so that they may be executed separately (lines 120, 130). Before storing the first screen (line 460) the message printed on line 21 must be erased; unless there is a special reason for retaining this. Removal is accomplished in line 450.

```
 10 REM STORE and RECALL
 15 REM DISPLAY and ATTRIBUTES
 20 REM set colours
 30 PAPER 6: INK 1: BORDER 6
 40 CLS
100 REM set new ramtop
110 CLEAR 58367: REM 16K=25599
120 LET start1=65281: REM 16K=3
2513
130 LET start2=65294: REM 16K=3
```

Program 6.1 *(continues)*

```
2526
 140 LET screen2=58368: REM 16K=
25600
 200 REM place routine
 210 LET length=13+12 .
 220 LET s=start1
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 280 REM point to second screen
 290 POKE 23296,screen2-256*INT
(screen2/256): POKE 23297,INT (s
creen2/256)
 300 REM create 1st triangle
 310 PLOT 38,50: DRAW 180,0: DRA
W -90,90: DRAW -90,-90
 400 REM store on keypress
 410 PRINT AT 21,0;"Press a key
to store"
 420 LET k$=INKEY$
 430 IF k$="" THEN  GO TO 420
 440 REM erase message
 450 PRINT AT 21,0;"
        "
 460 RANDOMIZE USR start1: REM s
tore
 500 REM create 2nd triangle
 510 PAPER 3: INK 7: BORDER 1: C
LS
 520 PRINT AT 21,0;"Press a key
to draw new triangle"
 530 LET k$=INKEY$
 540 IF k$="" THEN  GO TO 520
 550 CLS
 560 PLOT 38,120: DRAW 180,0: DR
AW -90,-90: DRAW -90,90
 600 REM recall 1st triangle
 610 PRINT AT 21,0;"Press a key
to recall 1st screen"
 620 LET k$=INKEY$
 630 IF k$="" THEN  GO TO 620
 700 REM recall 1st screen
 710 RANDOMIZE USR start2
 800 REM end
1000 REM STORE DISPLAY AND ATTRI
BUTES
1010 DATA 237,91,0,91,33,0,64,1,
0,27,237,176,201
2000 REM RECALL DISPLAY AND ATTR
IBUTES
2010 DATA 42,0,91,17,0,64,1,0,27
,237,176,201
```

Program 6.1

Storing and recalling attributes only allows the attributes of a picture to be changed rapidly without changing the contents of the display file. Since a second attribute file does not occupy a very large amount of memory, several attribute files may be held in memory, and the contents of the normal file changed as required.

Example 6.2

```
  10 REM STORE and RECALL
  15 REM ATTRIBUTES
  20 REM set colours
  30 PAPER 0: INK 7: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 58367: REM 16K=25599
 120 LET start1=65281: REM 16K=3
2513
 130 LET start2=65294: REM 16K=3
2526
 140 LET screen2=58368: REM 16K=
25600
 200 REM place routine
 210 LET length=13+12
 220 LET s=start1
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 280 REM point to second attribu
te file
 290 POKE 23296,screen2-256*INT
(screen2/256): POKE 23297,INT (s
creen2/256)
 300 REM create 1st triangle
 310 PLOT 38,50: DRAW 180,0: DRA
W -90,90: DRAW -90,-90
 400 REM store attribute file
 410 PRINT AT 21,0;"Press key to
 store attributes"
 420 LET k$=INKEY$
 430 IF k$="" THEN  GO TO 420
 460 RANDOMIZE USR start1: REM s
tore
 500 REM create 2nd triangle
 510 PAPER 3: INK 7: CLS
 520 PRINT AT 21,0;"Press a key
to draw new triangle"
 530 LET k$=INKEY$
 540 IF k$="" THEN  GO TO 530
 550 CLS
 560 PLOT 38,120: DRAW 180,0: DR
AW -90,-90: DRAW -90,90
```

Program 6.2 *(continues)*

```
 600 REM recall stored attribute
s
 610 PRINT AT 21,0;"Press key to
 recall attributes"
 620 LET k$=INKEY$
 630 IF k$="" THEN  GO TO 620
 640 REM erase message
 650 PRINT AT 21,0;"
                   "
 700 REM recall attribute file
 710 RANDOMIZE USR start2
 720 PAUSE 255
 800 REM end
1000 REM STORE ATTRIBUTES
1010 DATA 237,91,0,91,33,0,88,1,
0,3,237,176,201
2000 REM RECALL ATTRIBUTES
2010 DATA 42,0,91,17,0,88,1,0,3,
237,176,201
```

Program 6.2

Swapping screens

The contents of the visible and second screens may be exchanged using a short routine. Display and attribute files may be swapped together or separately. Screen swapping allows the currently-visible screen to be preserved since it is copied into the second screen area as the second screen is copied into the display file. Using this technique two screens can be constantly swapped, updated, and swapped again, providing a way of producing a repeatedly-changing display.

SWAP DISPLAY AND ATTRIBUTES

start address → Exchange the contents of the display and attribute files with the contents of a second set of display and attribute files

LENGTH 28 bytes

Using the routine

POKE the start address of the second screen into locations 23296 and 23297.

Swap display and attributes data

237, 91, 0, 91, 33, 0, 64, 1, 0, 27,
26, 245, 126, 18, 241, 119, 62, 0, 35, 19,
11, 184, 32, 242, 185, 32, 239, 201

Of course, there may be occasions when it is neither necessary nor desirable that both the display and attribute files are swapped. In that case, it is easy to swap the display files alone, or the attribute files alone.

SWAP DISPLAY

start address → Exchange the contents of the display file and a second display file

LENGTH 28 bytes

Using the routine

POKE the start address of the second display file into locations 23296 and 23297.

Swap display data

237, 91, 0, 91, 33, 0, 64, 1, 0, 24,
26, 245, 126, 18, 241, 119, 62, 0, 35, 19,
11, 184, 32, 242, 185, 32, 239, 201

SWAP ATTRIBUTES

start address → Exchange the contents of the attribute file and a second attribute file

LENGTH 28 bytes

Using the routine

POKE the start address of the second attribute file into locations 23296 and 23297.

Swap attributes data

237, 91, 0, 91, 33, 0, 88, 1, 0, 3,
26, 245, 126, 18, 241, 119, 62, 0, 35, 19,
11, 184, 32, 242, 185, 32, 239, 201

Example 6.3

This example shows how to use the routine which swaps display and attribute files.

```
  10 REM SWAP DISPLAY FILES
  20 REM set colours
  30 PAPER 6: INK 1: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 58367: REM 16K=25599
 120 LET start1=65281: REM 16K=3
2513
 130 LET start2=65294: REM 16K=3
2526
 140 LET start3=65306: REM 16K=3
2538
 150 LET screen2=58368: REM 16K=
25600
 200 REM place routine
 210 LET length=13+12+28
 220 LET s=start1
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 280 REM point to second screen
 290 POKE 23296,screen2-256*INT
(screen2/256): POKE 23297,INT (s
creen2/256)
 300 REM create 1st triangle
 310 PLOT 38,50: DRAW 180,0: DRA
W -90,90: DRAW -90,-90
 400 REM store on keypress
 410 PRINT AT 21,0;"Press a key
to store"
 420 LET k$=INKEY$
 430 IF k$="" THEN  GO TO 420
 440 REM erase message
 450 PRINT AT 21,0;"
                    "
 460 RANDOMIZE USR start1: REM s
tore
 500 REM create 2nd triangle
```



```
 510 PAPER 3: INK 7: CLS
 520 PRINT AT 21,0;"Press a key
to draw new triangle"
 530 LET k$=INKEY$
 540 IF k$="" THEN  GO TO 520
 550 CLS
 560 PLOT 38,120: DRAW 180,0: DR
AW -90,-90: DRAW -90,90
 600 REM recall 1st triangle
 610 PRINT AT 21,0;"Press a key
to swap screens"
 620 LET k$=INKEY$
 630 IF k$="" THEN  GO TO 620
 640 PRINT AT 21,0;"
                    "
 650 REM swap display and attrib
utes
 660 RANDOMIZE USR start3
 700 REM check stored files
 710 PRINT AT 21,0;"Press a key
to see stored files"
 720 LET k$=INKEY$
 730 IF k$="" THEN  GO TO 720
 740 PRINT AT 21,0;"
                    "
 750 RANDOMIZE USR start2
 800 REM end
 810 PAPER 7: INK 0
1000 REM STORE DISPLAY AND ATTRI
BUTES
1010 DATA 237,91,0,91,33,0,64,1,
0,27,237,176,201
2000 REM RECALL DISPLAY AND ATTR
IBUTES
2010 DATA 42,0,91,17,0,64,1,0,27
,237,176,201
3000 REM SWAP DISPLAY AND ATTRIB
UTES
3010 DATA 237,91,0,91,33,0,64,1,
0,27
3020 DATA 26,245,126,18,241,119,
62,0,35,19
3030 DATA 11,184,32,242,185,32,2
39,201
```

Program 6.3

When the final message appears (see line 710) the PAPER and INK colours used will be the permanent colours set in line 510, and not those of the display and attribute files which have just been recalled. Try changing the values used in line 510, to see this effect more clearly.

Adding screens

Instead of swapping one screen for another, composite pictures may be formed by adding the contents of two screens. This can be done by adding the second screen onto the normal display file, or by reversing the process and adding the display file onto the second screen, which achieves the same end but performs the addition out of sight. In either case, the attributes are not added, since this would produce rather unpredictable results. Instead, the attributes used for the composite screen belong to the attribute file associated with the screen on which the addition is placed. So, if the second screen is added to the display file the attributes are those which already belong to the display file. If the display file is added to the second screen, the attributes are those of the second screen. If this does not produce the required effect, one of the other routines may be used to manipulate the attributes as necessary.

ADD STORE TO DISPLAY

start address →

Adds the contents of a second screen to the contents of the display file and stores the result on the display file

LENGTH 25 bytes

Using the routine

POKE the start address of the second screen into locations 23296 and 23297.

Add store to display data

237, 91, 0, 91, 33, 0, 64, 1, 0, 24,
26, 182, 119, 62, 0, 35, 19, 11, 184, 32,
245, 185, 32, 242, 201

ADD DISPLAY TO STORE

start address →

Adds the contents of the display file to the contents of a second screen and stores the result on the second screen

LENGTH 24 bytes

Using the routine

POKE the start address of the second screen into locations 23296 and 23297.

Add display to store data

42, 0, 91, 17, 0, 64, 1, 0, 24, 26,
182, 119, 62, 0, 35, 19, 11, 184, 32, 245,
185, 32, 242, 201

Clearing a second screen

From time to time it may be necessary to clear the second screen and/or attributes. In the case of the second display file this means setting all the bytes in that area to zero. The attribute file requires a rather different approach, however, since setting all the bytes in that area to zero would produce an attribute file in which the PAPER and INK colours were both set to zero, i.e., black.

The usual requirement is for PAPER 7 and INK 0, so the attribute clearing routine clears to this set of colours unless steps are taken to alter these.

Two separate routines are presented; one for each of the files in the second screen area. Clearing both the second display file and the second attribute file will require both of the routines to be executed, one after the other, in any order. In the case when only one of the areas needs to be cleared, simply use the appropriate routine on its own.

Note that when a second screen is created, there is no guarantee that the contents of the second display file will be 0. Similarly, there is no reason to suppose that the bytes in the second attribute file will be set to 56 (i.e., PAPER 7 : INK 0 : FLASH 0 : BRIGHT 0). Creating a second screen merely reserves an area of memory; it does not alter the previous contents of memory. So, when a second screen is created, it is good practice to clear the area before plotting on it. In the previous examples, it was not necessary to clear the second screen area before use, because transferring the first screen into the second screen area ensures that the contents of the second screen are known, and any previous contents will be destroyed in the process.

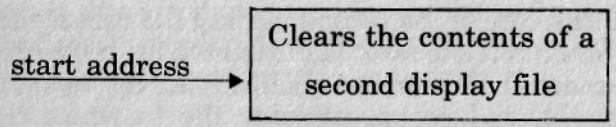
Another way to clear the second display area and the second attribute file is to

1. Set the visible attributes to the value required for the second attribute file.
2. Clear the visible display file.

3. Transfer the visible display and attribute files into the second screen area using the STORE DISPLAY AND ATTRIBUTES routine.

This has the advantage that the attributes may be set to a variety of values within the attribute file, but it suffers from the disadvantage that the process becomes visible, and is normally only suitable for use at a point in the program when the visible display file is being cleared anyway. However, it is sometimes useful for setting up a complex second attribute file at the start of a program, when the screen is often cleared for a few moments anyway.

CLEAR DISPLAY



LENGTH 18 bytes

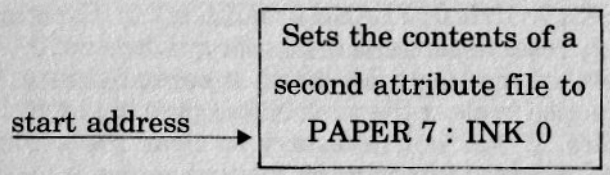
Using the routine

POKE the start address of the second display file into locations 23296 and 23297.

Clear display data

42, 0, 91, 1, 0, 24, 62, 0, 119, 35,
11, 184, 32, 249, 185, 32, 247, 201

CLEAR ATTRIBUTES



LENGTH 20 bytes

Using the routine

POKE the start address of the second attribute file into locations 23296 and 23297.

Note: The 8th item of data represents the contents of the attribute byte which is placed in each location in the second attribute file.

Any other suitable attribute value may be substituted. The routine will then clear the second attribute file to a specific set of attributes.

Clear attributes data

42, 0, 91, 1, 0, 3, 62, 56*, 119, 62,
0, 35, 11, 184, 32, 246, 185, 32, 243, 201

Example 6.4

```
 10 REM STORE,RECALL,ADD,CLEAR
 15 REM DISPLAY and ATTRIBUTES
 20 REM set colours
 30 PAPER 6: INK 1: BORDER 6
 40 CLS
100 REM set new ramtop
110 CLEAR 58367: REM 16K=25599
120 LET start1=65281: REM 16K=3
2513
130 LET start2=65294: REM 16K=3
2526
140 LET start3=65306: REM 16K=3
2538
150 LET start4=65331: REM 16K=3
2563
160 LET screen2=58368: REM 16K=
25600
170 LET as=screen2+6144: REM st
art of attribute store
200 REM place routine
210 LET length=13+12+25+20
220 LET s=start1
230 FOR i=1 TO length
240 READ n
250 POKE s,n
260 LET s=s+1
270 NEXT i
280 REM point to second screen
290 POKE 23296,screen2-256*INT
(screen2/256): POKE 23297,INT (s
creen2/256)
300 REM create 1st triangle
310 PLOT 38,50: DRAW 180,0: DRA
W -90,90: DRAW -90,-90
400 REM store on keypress
410 PRINT AT 21,0;"Press key to
store"
420 LET k$=INKEY$
430 IF k$="" THEN  GO TO 420
440 REM erase message
450 PRINT AT 21,0;"
  "
460 RANDOMIZE USR start1: REM s
tore
500 REM create 2nd triangle
```

Program 6.4 *(continues)*

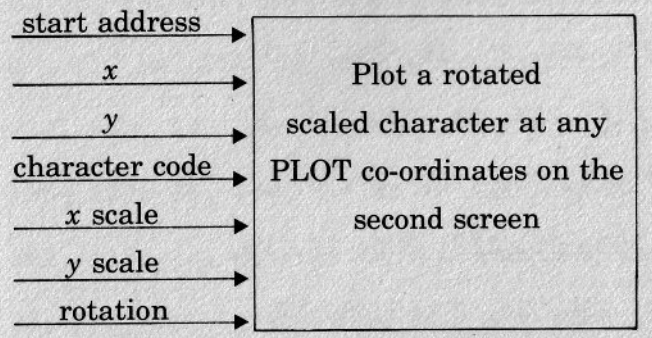
```
 510 PAPER 3: INK 7: BORDER 1: C
LS
 520 PRINT AT 21,0;"Press key to
 draw new triangle"
 530 LET k$=INKEY$
 540 IF k$="" THEN  GO TO 520
 550 CLS
 560 PLOT 38,120: DRAW 180,0: DR
AW -90,-90: DRAW -90,90
 600 REM add store to display
 610 PRINT AT 21,0;"Press key to
 add stored screen"
 620 LET k$=INKEY$
 630 IF k$="" THEN  GO TO 620
 640 RANDOMIZE USR start3
 650 REM erase message
 660 PRINT AT 21,0;"
                   "
 700 REM clear stored attributes
 710 REM point to stored attribu
tes
 720 POKE 23296,as-256*INT (as/2
56): POKE 23297,INT (as/256)
 730 RANDOMIZE USR start4
 740 REM point to stored display
 again
 750 POKE 23296,screen2-256*INT
(screen2/256): POKE 23297,INT (s
creen2/256)
 800 REM recall stored screen
 810 PRINT AT 21,0;"Press key to
 recall store"
 820 LET k$=INKEY$
 830 IF k$="" THEN  GO TO 720
 840 RANDOMIZE USR start2
1000 REM STORE DISPLAY AND ATTRI
BUTES
1010 DATA 237,91,0,91,33,0,64,1,
0,27,237,176,201
2000 REM RECALL DISPLAY AND ATTR
IBUTES
2010 DATA 42,0,91,17,0,64,1,0,27
,237,176,201
3000 REM ADD STORE TO DISPLAY
3010 DATA 237,91,0,91,33,0,64,1,
0,24
3020 DATA 26,182,119,62,0,35,19,
11,184,32
3030 DATA 245,185,32,242,201
4000 REM CLEAR STORED ATTRIBUTES
4010 DATA 42,0,91,1,0,3,62,56,11
9,62
4020 DATA 0,35,11,184,32,246,185
,32,243,201
```

Program 6.4

Printing

Very often, it is desirable to be able to print characters on a display file while it is held in the second screen area, out of sight. This allows changes to be made 'invisibly', and then revealed as the second screen is swapped into the main display file area. The most useful routine for printing characters is based on the ROTATED SCALED CHARACTER PLOTTER routine detailed in Chapter 1.

SECOND SCREEN CHARACTER PLOTTER



LENGTH 414 bytes

Using the routine

POKE the start address of the second screen into locations 23296 and 23297.
POKE the *x* co-ordinate of the bottom left corner of the character into location 23298.
POKE the *y* co-ordinate of the bottom left corner of the character into location 23299.
POKE the character code into location 23300.
POKE the *x* scale into location 23301.
POKE the *y* scale into location 23302.
POKE the rotation into location 23303.

Note: The rotation must be in the range 0–3. The routine uses locations 23304–23315 inclusive, as workspace. User-defined characters may be plotted.

Second screen character plotter data

```
58, 7, 91, 254, 4, 48, 43, 58, 4, 91,

254, 32, 56, 36, 254, 128, 56, 8, 254, 144,
```

```
56, 28, 254, 165, 56, 7, 33, 7, 61, 214,
32, 24, 6, 42, 123, 92, 43, 214, 143, 17,
8, 0, 254, 0, 40, 6, 25, 61, 24, 248,
24, 104, 58, 7, 91, 254, 0, 32, 10, 17,
19, 91, 1, 8, 0, 237, 184, 24, 82, 254,
2, 32, 25, 6, 8, 17, 10, 91, 197, 26
79, 126, 6, 8, 23, 203, 25, 16, 251, 121
18, 193, 19, 43, 16, 238, 24, 53, 254, 3,
32, 25, 6, 8, 17, 19, 91, 126, 197, 6,
8, 23, 235, 78, 203, 25, 113, 235, 27, 16,
246, 193, 43, 16, 235, 24, 24, 6, 8, 17,
19, 91, 126, 197, 6, 8, 203, 31, 235, 78,
203, 17, 113, 235, 27, 16, 245, 193, 43, 16,
234, 33, 19, 91, 229, 24, 2, 24, 70, 33,
2, 91, 126, 79, 35, 126, 71, 33, 6, 91,
126, 35, 35, 35, 119, 35, 35, 62, 0, 119,
225, 24, 23, 241, 193, 225, 43, 229, 33, 6,
91, 126, 35, 35, 35, 119, 225, 24, 3, 241,
193, 225, 58, 2, 91, 79, 126, 229, 197, 245,
33, 10, 91, 62, 0, 119, 241, 7, 245, 33,
5, 91, 126, 35, 35, 35, 119, 24, 8, 24,
103, 24, 206, 24, 220, 24, 235, 62, 16, 128,
```



```
71, 42, 0, 91, 254, 128, 48, 9, 17, 0,
8, 25, 254, 64, 48, 1, 25, 203, 184, 203,
176, 62, 63, 144, 17, 32, 0, 254, 8, 56,
5, 214, 8, 25, 24, 247, 254, 0, 40, 4,
61, 36, 24, 248, 121, 254, 8, 56, 5, 214,
8, 35, 24, 247, 79, 62, 7, 145, 55, 63,
87, 71, 94, 254, 0, 40, 4, 203, 11, 16,
252, 241, 245, 56, 4, 203, 131, 24, 2, 203,
195, 122, 66, 254, 0, 40, 4, 203, 3, 16,
252, 115, 24, 8, 24, 77, 24, 149, 24, 149,
24, 149, 33, 8, 91, 126, 61, 119, 254, 0,
40, 7, 241, 193, 12, 197, 245, 24, 134, 33,
10, 91, 126, 60, 119, 254, 8, 40, 7, 241,
193, 12, 197, 245, 24, 220, 33, 9, 91, 126,
61, 119, 254, 0, 40, 7, 241, 193, 4, 197,
245, 24, 201, 33, 11, 91, 126, 60, 119, 254,
8, 40, 7, 241, 193, 4, 197, 245, 24, 182,
241, 193, 225, 201
```

Example 6.5

```
  10 REM WRITING ON A SECOND SCR
EEN
  20 REM set colours
  30 PAPER 7: INK 0: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 57856: REM 16K=25088
 120 LET start1=64769: REM 16K=3
```

Program 6.5 *(continues)*

```
2001
 130 LET start2=65183: REM 16K=3
2415
 140 LET start3=65201: REM 16K=3
2433
 150 LET start4=65221: REM 16K=3
2453
 160 LET screen2=57857: REM 16K=
25089
 200 REM place routines
 210 LET length=414+18+20+12
 220 LET s=start1
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 280 REM point to second screen
 290 POKE 23296,screen2-256*INT
(screen2/256): POKE 23297,INT (s
creen2/256)
 300 REM clear second screen
 310 RANDOMIZE USR start2
 320 REM clear second attribute
file
 330 REM point to second attribu
te file
 340 POKE 23296,screen2+6144-256
*INT ((screen2+6144)/256): POKE
23297,INT ((screen2+6144)/256)
 350 RANDOMIZE USR start3
 360 REM point to second display
 file again
 370 POKE 23296,screen2-256*INT
(screen2/256): POKE 23297,INT (s
creen2/256)
 400 REM input word to be printe
d on second screen
 410 PRINT "Please type your fir
st name"
 420 INPUT n$
 430 IF LEN n$>31 THEN  PRINT "S
orry...that name is too long": P
RINT "for this demonstration.":
PRINT "Please try a shorter name
.": GO TO 410
 500 REM print word
 510 POKE 23298,0: REM x co-ord
 520 POKE 23299,50: REM y co-ord
 530 POKE 23301,1: REM x scale
 540 POKE 23302,1: REM y scale
 550 POKE 23303,0: REM rotation
 560 FOR i=1 TO LEN n$
 570 POKE 23300,CODE n$(i)
 580 RANDOMIZE USR start1
```

Program 6.5 *(continues)*


```
 590 POKE 23298,PEEK (23298)+8
 600 NEXT i
 700 REM show result
 710 CLS
 720 PRINT "Press a key to see t
he second": PRINT "screen."
 730 LET k$=INKEY$
 740 IF k$="" THEN  GO TO 730
 750 RANDOMIZE USR start4
 800 REM finished
1000 REM PLOT CHARACTER ON
1005 REM SECOND SCREEN
1010 DATA 58,7,91,254,4,48,43,58
,4,91
1020 DATA 254,32,56,36,254,128,5
6,8,254,144
1030 DATA 56,28,254,165,56,7,33,
7,61,214
1040 DATA 32,24,6,42,123,92,43,2
14,143,17
1050 DATA 8,0,254,0,40,6,25,61,2
4,248
1060 DATA 24,104,58,7,91,254,0,3
2,10,17
1070 DATA 19,91,1,8,0,237,184,24
,82,254
1080 DATA 2,32,25,6,8,17,10,91,1
97,26
1090 DATA 79,126,6,8,23,203,25,1
6,251,121
1100 DATA 18,193,19,43,16,238,24
,53,254,3
1110 DATA 32,25,6,8,17,19,91,126
,197,6
1120 DATA 8,23,235,78,203,25,113
,235,27,16
1130 DATA 246,193,43,16,235,24,2
4,6,8,17
1140 DATA 19,91,126,197,6,8,203,
31,235,78
1150 DATA 203,17,113,235,27,16,2
45,193,43,16
1160 DATA 234,33,19,91,229,24,2,
24,70,33
1170 DATA 2,91,126,79,35,126,71,
33,6,91
1180 DATA 126,35,35,35,119,35,35
,62,0,119
1190 DATA 225,24,23,241,193,225,
43,229,33,6
1200 DATA 91,126,35,35,35,119,22
5,24,3,241
1210 DATA 193,225,58,2,91,79,126
,229,197,245
1220 DATA 33,10,91.62,0,119,241,
```

```
7,245,33
1230 DATA 5,91,126,35,35,35,119,
24,8,24
1240 DATA 103,24,206,24,220,24,2
35,62,16,128
1250 DATA 71,42,0,91,254,128,48,
9,17,0
1260 DATA 8,25,254,64,48,1,25,20
3,184,203
1270 DATA 176,62,63,144,17,32,0,
254,8,56
1280 DATA 5,214,8,25,24,247,254,
0,40,4
1290 DATA 61,36,24,248,121,254,8
,56,5,214
1300 DATA 8,35,24,247,79,62,7,14
5,55,63
1310 DATA 87,71,94,254,0,40,4,20
3,11,16
1320 DATA 252,241,245,56,4,203,1
31,24,2,203
1330 DATA 195,122,66,254,0,40,4,
203,3,16
1340 DATA 252,115,24,8,24,77,24,
149,24,149
1350 DATA 24,149,33,8,91,126,61,
119,254,0
1360 DATA 40,7,241,193,12,197,24
5,24,134,33
1370 DATA 10,91,126,60,119,254,8
,40,7,241
1380 DATA 193,12,197,245,24,220,
33,9,91,126
1390 DATA 61,119,254,0,40,7,241,
193,4,197
1400 DATA 245,24,201,33,11,91,12
6,60,119,254
1410 DATA 8,40,7,241,193,4,197,2
45,24,182
1420 DATA 241,193,225,201
2000 REM CLEAR DISPLAY
2010 DATA 42,0,91,1,0,24,62,0,11
9,35
2020 DATA 11,184,32,249,185,32,2
47,201
3000 REM CLEAR ATTRIBUTES
3010 DATA 42,0,91,1,0,3,62,56,11
9,62
3020 DATA 0,35,11,184,32,246,185
,32,243,201
4000 REM RECALL DISPLAY AND ATTR
IBUTES
4010 DATA 42,0,91,17,0,64,1,0,27
,237,176,201
```

Program 6.5

Four routines are used in this demonstration, and the start addresses START1 to START4 refer to the SECOND SCREEN CHARACTER PLOTTER, CLEAR DISPLAY, CLEAR ATTRIBUTES, and RECALL DISPLAY AND ATTRIBUTES routines. Line 590 increments the x co-ordinate between plotting letters. This is done by examining the current x co-ordinate and adding 8. The spacing between letters can be varied by changing the 8 for some other value. The initial value of the x co-ordinate is set to 0 in line 510, and this may be changed to start plotting at some other, more convenient, value.

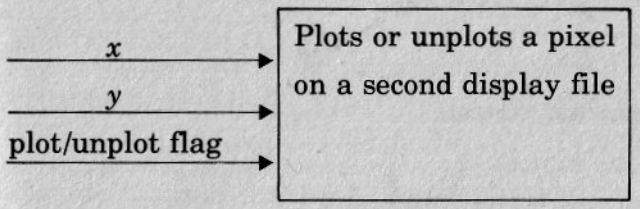
Clearing the second attribute file demands that the contents of locations 23296 and 23297 represent the start address of the second attribute file. This is set up in line 340, and then reset to point to the start of the second screen in line 370.

Experiment with the initial values of x and y, the increments between letters, and the value of rotation, to gain an understanding of the process of plotting characters on the second screen.

Drawing

Drawing can also take place on a second screen, by using a routine which plots or unplots pixels. This extends the use of a second screen quite considerably. The routine also incorporates a collision flag which is set to 1 when plotting takes place at a position at which a pixel is already illuminated.

SECOND SCREEN PIXEL PLOTTER



LENGTH 118 bytes

Using the routine

POKE the start address of the second screen into locations 23296 and 23297.

POKE the x co-ordinate of the pixel into location 23298.

POKE the y co-ordinate of the pixel into location 23299.

POKE 1 (to plot) or 0 (to unplot) into location 23301.

Note: The routine returns with a 1 in location 23302 if the pixel to be plotted was already ON.

If this collision flag is to be used, it should be set to 0 initially, by POKEing 0 into location 23302, before executing the routine.

Second screen pixel plotter data

```
33, 2, 91, 78, 35, 70, 62, 16, 128, 71,

42, 0, 91, 254, 128, 48, 9, 17, 0, 8,

25, 254, 64, 48, 1, 25, 203, 184, 203, 176,

62, 63, 144, 17, 32, 0, 254, 8, 56, 5,

214, 8, 25, 24, 247, 254, 0, 40, 4, 61,

36, 24, 248, 121, 254, 8, 56, 5, 214, 8,

35, 24, 247, 79, 62, 7, 145, 55, 63, 87,

71, 94, 254, 0, 40, 4, 203, 11, 16, 252,

203, 67, 40, 5, 62, 1, 50, 6, 91, 241,

245, 48, 13, 58, 5, 91, 254, 1, 32, 4,

203, 195, 24, 2, 203, 131, 122, 66, 254, 0,

40, 4, 203, 3, 16, 252, 115, 201
```

Example 6.6

```
  10 REM DRAW ON SECOND SCREEN
  20 REM set colours
  30 PAPER 7: INK 0: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 58367: REM 16K=25599
 120 LET start1=65281: REM 16K=3
2513
 130 LET start2=65399: REM 16K=3
2631
 140 LET start3=65417: REM 16K=3
2649
 150 LET screen2=58368: REM 16K=
25600
 200 REM place routines
```

Program 6.6 *(continues)*


```
 210 LET length=118+18+25
 220 LET s=start1
 230 FOR i=1 TO length
 240 READ n
 250 POKE s,n
 260 LET s=s+1
 270 NEXT i
 280 REM point to second screen
 290 POKE 23296,screen2-256*INT
(screen2/256): POKE 23297,INT (s
creen2/256)
 300 REM draw on visible screen
 310 CIRCLE 128,86,20
 400 REM clear second screen
 405 REM then draw rectangle
 410 RANDOMIZE USR start2
 420 POKE 23301,1: REM plot
 430 POKE 23299,66: REM y co-ord
 440 FOR x=108 TO 148
 450 POKE 23298,x
 460 RANDOMIZE USR start1
 470 NEXT x
 480 FOR y=66 TO 106
 490 POKE 23299,y
 500 RANDOMIZE USR start1
 510 NEXT y
 520 FOR x=148 TO 108 STEP -1
 530 POKE 23298,x
 540 RANDOMIZE USR start1
 550 NEXT x
 560 FOR y=106 TO 66 STEP -1
 570 POKE 23299,y
 580 RANDOMIZE USR start1
 590 NEXT y
 600 REM add the screens
 610 PRINT AT 21,0;"Press a key
to add the screens."
 620 LET k$=INKEY$
 630 IF k$="" THEN  GO TO 620
 640 PRINT AT 21,0;"
                    "
 650 RANDOMIZE USR start3
 700 REM finished
1000 REM SECOND SCREEN PIXEL PLO
TTER
1010 DATA 33,2,91,78,35,70,62,16
,128,71
1020 DATA 42,0,91,254,128,48,9,1
7,0,8
1030 DATA 25,254,64,48,1,25,203,
184,203,176
1040 DATA 62,63,144,17,32,0,254,
8,56,5
1050 DATA 214,8,25,24,247,254,0,
40,4,61
```

```
1060 DATA 36,24,248,121,254,8,56
,5,214,8
1070 DATA 35,24,247,79,62,7,145,
55,63,87
1080 DATA 71,94,254,0,40,4,203,1
1,16,252
1090 DATA 203,67,40,5,62,1,50,6,
91,241
1100 DATA 245,48,13,58,5,91,254,
1,32,4
1110 DATA 203,195,24,2,203,131,1
22,66,254,0
1120 DATA 40,4,203,3,16,252,115,
201
2000 REM CLEAR DISPLAY
2010 DATA 42,0,91,1,0,24,62,0,11
9,35
2020 DATA 11,184,32,249,185,32,2
47,201
3000 REM ADD STORE TO DISPLAY
3010 DATA 237,91,0,91,33,0,64,1,
0,24
3020 DATA 26,182,119,62,0,35,19,
11,184,32
3030 DATA 245,185,32,242,201
```

Program 6.6

In this example, the second display file must be cleared before drawing takes place. This is done by using the CLEAR DISPLAY routine, in line 410. The plotting routine is then set up and the x and y co-ordinates are indexed to produce the outline of a square. The final result is produced by adding the second display area onto the normally-visible display file, using the ADD STORE TO DISPLAY routine, in line 650. Line 640 consists of 31 SPACE characters, sufficient to erase the message printed in line 610.

Preserving screens during addition

In the last example, the final result was produced by adding the second display file to the first. This left the first display file holding a copy of the sum of the two display files, and the second display file was left intact. Suppose the required result was:
the sum of the contents of the two display files in the visible display file, and
a copy of the first, visible, display file in the second display file.

This could be achieved by:
(a) adding the first display file onto the second display file, using the ADD DISPLAY TO STORE routine.
(b) swapping the first and second display files, using the SWAP DISPLAY routine.

This would produce the required effect, preserving the contents of the first display file in the process.

Other screens

Using exactly the same techniques as for a second screen, additional screens may be held in memory, out of sight.

Separate display and attribute files may be used if necessary, so that there could be a second display file and (say) five attribute files. Just remember to POKE the start address of the file to be used into locations 23296 and 23297 before manipulating the contents of that file.

Since the routines pick up the address of the display and attribute areas from 23296 and 23297, any of the second screen routines may be used with the normally-visible display and attribute files. The start addresses should be placed in 23296 and 23297 as for the other areas. This is sometimes useful if characters are being plotted at similar x or y co-ordinates on one screen after another, since some of the set values may be retained.

Saving and loading screens

It is very useful to be able to create a picture using one program, and then save it to tape so that it can be recalled by another program in which it is to be used.

The commands

```
SAVE "name" SCREEN$
```

and

```
SAVE "name" CODE 16384,6912
```

are equivalent, and provide a way of saving a copy of the screen to tape.

```
SAVE "name" CODE
```

saves the bytes in a section of memory to tape, and should be followed by an address at which to start, and a length. Since the display file begins at 16384, saving 6912 bytes beginning at that location saves

(a) a copy of the condition of each pixel on the screen (including the bottom 16 lines of 32 bytes) – 6144 bytes altogether.
(b) a copy of the attribute file – 768 bytes

Sometimes it is necessary to save the display file alone, or the attribute file alone.

SAVE "name" CODE 16384,6144 saves only the display file, while
SAVE "name" CODE 22528,768 saves only the attribute file.

Loading a copy of the display and attribute files from tape requires the reverse operation.

LOAD "name" SCREEN$

or

LOAD "name" CODE 16384,6912

loads a copy of the display and attributes files.

LOAD "name" CODE 16384,6144 loads copy of the display file.
LOAD "name" CODE 22528,768 loads copy of the attribute file.

If the command

LOAD "name" CODE

is used, without specifying the start address and the length, the Spectrum will look at the information on the tape and note where it came from originally. (This information is stored in the header.) It will then load the data into the section of memory from which it was originally saved. The display file will be loaded first, followed by the attributes (assuming both files have been stored). This explains why the colours appear on the screen after the basic shapes. If only one of the files has been stored, then only that file will be loaded.

Considerable flexibility is possible, using SAVE and LOAD with CODE along with the second screen routines.

A second screen can be saved to tape simply by using appropriate start addresses for the files which are to be saved. The start addresses will be those already used in the program to point to the start of the second display and attribute files.

When loading a second screen from tape, suitable addresses should be used to specify where the information should begin loading. The files may load into the normally-visible screen and attribute areas, or they may be directed into a second screen area, in which case they will load without being seen. They may then be switched into the visible file areas as required, using some of the

routines already given for this purpose. Simple, but very effective, slide shows are possible which repeatedly load copies of a second screen and display them once they have loaded.

An effective video titling program can be produced using this method. The first program on a tape may be designed to load a display file into a second screen area, then display it, and then scroll it (e.g., scrolling the title and credits for the film). The next file is then loaded and treated in a similar fashion . . . and so on.

Preserving a background

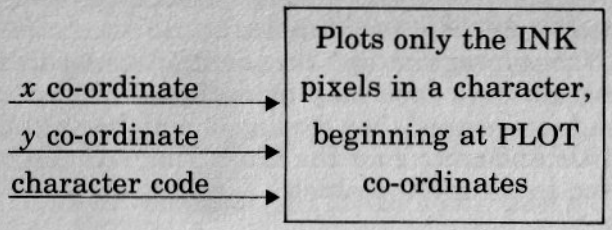
A second screen can be used to provide a constant background scene, by repeatedly refreshing the display from a stored screen. Because the RECALL DISPLAY routine works very quickly, it is easy to move shapes around the screen without appearing to interfere with the background. This really does provide beautiful, flicker-free, arcade-quality graphics.

The method is quite straightforward:

Create the background
Store a copy of the background in a second screen.
Set the co-ordinates at which to plot.
Plot a character on the main screen at the current position.
Update the character position.
Recall the background.

Using this method, a fresh copy of the background is placed on the main screen before the characters are printed, so any trace of the previous characters is erased before further characters are added to the foreground. The whole process is very effective.

However, one small problem remains. If characters are printed on the screen at PRINT positions, OVER1 can be set to add the lit pixels in the character to the background. If this is not done, there is a danger that the PAPER in the 8 × 8 charactter block will erase some of the detail in the background. If more accurate positioning is needed, and characters are being plotted at PLOT positions, a new routine can be used to plot only the INK pixels in the character. This guarantees that the character will be added to the screen in the expected way, with no unnecessary interference to the background.



LENGTH 214 bytes

Using the routine

POKE the *x* co-ordinate of the bottom left corner of the character into location 23298.
POKE the *y* co-ordinate of the bottom left corner of the character into location 23299.
POKE the character code into location 23300.

Note: A collision flag is provided in location 23301. This flag is automatically set to 0 when the routine begins. After plotting a character, this flag may be examined. Its contents will be:

0 if there were no INK pixels already plotted under any of the INK pixels in the character just plotted.
1 if there was at least one INK pixel already plotted under any of the INK pixels in the character just plotted.

This routine can be made to plot on a second screen, by loading the routine into memory and making the following changes:

Change the 102nd item of data (currently 33) to 42
Change the 104th item of data (currently 64) to 91

The address of the second screen should be placed in locations 23296, and 23297. As shown in the next example, this address is normally held in those locations anyway, if this routine is being used in conjunction with one of the second screen routines. The changes to the data items can be made once the routine has been loaded, using BASIC commands.
If start denotes the address of the start of the routine, use:

```
POKE start+101,42
POKE start+103,91
```

Paperless plotter data

```
62, 0, 50, 5, 91, 58, 4, 91, 254, 32,

56, 81, 254, 128, 56, 10, 254, 144, 56, 73,
```



```
254, 165, 56, 9, 24, 67, 33, 7, 61, 214,

32, 24, 6, 42, 123, 92, 43, 214, 143, 17,

8, 0, 254, 0, 40, 4, 25, 61, 24, 248,

6, 0, 126, 43, 229, 14, 0, 24, 1, 241,

7, 245, 197, 56, 2, 24, 24, 33, 2, 91,

126, 254, 249, 48, 20, 129, 79, 35, 126, 254,

169, 48, 12, 128, 71, 24, 10, 24, 226, 24,

217, 24, 100, 24, 118, 24, 113, 62, 16, 128,

71, 33, 0, 64, 254, 128, 48, 9, 17, 0,

8, 25, 254, 64, 48, 1, 25, 203, 184, 203,

176, 62, 63, 144, 17, 32, 0, 254, 8, 56,

5, 214, 8, 25, 24, 247, 254, 0, 40, 4,

61, 36, 24, 248, 121, 254, 8, 56, 5, 214,

8, 35, 24, 247, 79, 62, 7, 145, 55, 63,

87, 71, 94, 254, 0, 40, 4, 203, 11, 16,

252, 203, 67, 40, 5, 62, 1, 50, 5, 91,

203, 195, 122, 66, 254, 0, 40, 4, 203, 3,

16, 252, 115, 193, 12, 121, 254, 8, 32, 143,

4, 241, 225, 120, 254, 8, 32, 137, 40, 3,

193, 241, 225, 201
```

Example 6.7

CONSTANT BACKGROUND provides just that; an apparently constant and unchanging background view, in front of which a little

bird flies (actually, it's that gnat from Chapter 4, but it flaps its wings just like a little bird, so it will do fine!)

```
 10 REM CONSTANT BACKGROUND
 20 CLS
100 REM set new ramtop
110 CLEAR 58879
120 LET start=65127
200 REM load routines
210 LET length=214+13+12
220 LET s=start
230 FOR i=1 TO length
240 READ n
250 POKE s,n
260 LET s=s+1
270 NEXT i
300 LET screen2=58880
310 LET cplot=start
320 LET store=cplot+214
330 LET recall=store+13
400 REM define UDG characters
410 FOR i=1 TO 9
420 READ c$
430 FOR j=0 TO 7
440 READ n
450 POKE USR c$+j,n
460 NEXT j
470 NEXT i
500 REM create background
510 PAPER 5: INK 4: CLS
520 PAPER 4: INK 4
530 FOR l=20 TO 21
540 FOR c=0 TO 31
550 PRINT AT l,c;" "
560 NEXT c
570 NEXT l
580 PAPER 5: INK 4
590 LET l=16
600 FOR c=5 TO 25 STEP 5
610 PRINT AT l,c;CHR$ 146+CHR$
147+CHR$ 148
620 PRINT AT l+1,c;CHR$ 149+CHR
$ 150+CHR$ 151
630 PRINT AT l+2,c+1;CHR$ 152
640 PRINT AT l+3,c+1;CHR$ 152
650 NEXT c
700 REM store background
710 POKE 23296,screen2-256*INT
(screen2/256)
720 POKE 23297,INT (screen2/256
)
730 RANDOMIZE USR store
740 LET d=0
750 LET y=35
```

Program 6.7 *(continues)*


```
1000 REM main loop
1010 FOR i=1 TO 4: REM no. of pa
sses
1020 FOR x=248 TO 0 STEP -2
1030 POKE 23298,x: POKE 23299,y:
 POKE 23300,144+d: RANDOMIZE USR
 cplot
1040 LET d=1-d
1050 RANDOMIZE USR recall
1060 NEXT x
1070 NEXT i
1100 REM restore colours
1110 PAPER 7: INK 0: BORDER 7
2000 REM CHARACTER PLOTTER
2010 DATA 62,0,50,5,91,58,4,91,2
54,32
2020 DATA 56,81,254,128,56,10,25
4,144,56,73
2030 DATA 254,165,56,9,24,67,33,
7,61,214
2040 DATA 32,24,6,42,123,92,43,2
14,143,17
2050 DATA 8,0,254,0,40,4,25,61,2
4,248
2060 DATA 6,0,126,43,229,14,0,24
,1,241
2070 DATA 7,245,197,56,2,24,24,3
3,2,91
2080 DATA 126,254,249,48,20,129,
79,35,126,254
2090 DATA 169,48,12,128,71,24,10
,24,226,24
2100 DATA 217,24,100,24,118,24,1
13,62,16,128
2110 DATA 71,33,0,64,254,128,48,
9,17,0
2120 DATA 8,25,254,64,48,1,25,20
3,184,203
2130 DATA 176,62,63,144,17,32,0,
254,8,56
2140 DATA 5,214,8,25,24,247,254,
0,40,4
2150 DATA 61,36,24,248,121,254,8
,56,5,214
2160 DATA 8,35,24,247,79,62,7,14
5,55,63
2170 DATA 87,71,94,254,0,40,4,20
3,11,16
2180 DATA 252,203,67,40,5,62,1,5
0,5,91
2190 DATA 203,195,122,66,254,0,4
0,4,203,3
2200 DATA 16,252,115,193,12,121,
254,8,32,143
2210 DATA 4,241,225,120,254,8,32
```

```
,137,40,3
2220 DATA 193,241,225,201
3000 REM STORE DISPLAY
3010 DATA 237,91,0,91,33,0,64,1,
0,24
3020 DATA 237,176,201
4000 REM RECALL DISPLAY
4010 DATA 42,0,91,17,0,64,1,0,24
,237
4020 DATA 176,201
5000 REM UDG DEFINITIONS
5010 DATA "A",0,0,32,126,28,28,8
,0
5020 DATA "B",0,8,60,126,0,0,0,0
5030 DATA "C",0,0,3,3,15,31,59,5
9
5040 DATA "D",30,254,255,255,255
,255,255,255
5050 DATA "E",0,0,128,224,224,22
4,248,248
5060 DATA "F",51,119,116,100,0,0
,0,0
5070 DATA "G",127,127,127,127,60
,60,60,60
5080 DATA "H",88,88,72,8,0,0,0,0
5090 DATA "I",60,60,60,60,60,60,
60,60
```

Program 6.7

Three routines are used:

PAPERLESS PLOTTER
STORE DISPLAY
RECALL DISPLAY

The background is created by lines 500–650, and is then stored in a second screen (lines 700–730).

Note lines 710 and 720, which define the start address of the second screen.

In line 740, *d* represents a quantity which will alternate between 0 and 1 (see line 1040) and which will be added to 144 to give character codes 144 or 145, alternatively, to select the characters to be plotted. In line 750, the height at which the little bird flies is fixed. Lines 1020–1060 repeatedly plot the bird, then restore the background. Notice too, that because restoring the background erases the bird, it is no longer necessary to restrict movement between plotting positions to the number of blank pixels incorporated at the trailing edge of the character. Try varying the step size in line 1020 to see the effect.

The use of the collision flag in location 23301 can be illustrated by adding line 1035, like this:

```
1035 IF PEEK (23301) = THEN PRINT INK 0; AT 0,0;"OUCH!"
:STOP
```

7 SCROLLING

Speed is an essential ingredient in many programs. In particular, moving large sections of the screen quickly can cause all sorts of headaches when programming. Scrolling, in its many forms, often provides the remedy, since it is an ideal way of handling the movement of a background, and adds significantly to the quality of a graphics display.

This chapter presents a large number of routines for scrolling the display file and/or the attribute file. By the end of this chapter, the problem should not be how to scroll the screen, but which of the many routines will scroll the screen in the desired fashion! Scrolling of the display file can be performed in any of four directions: left, right, up, or down.

In each direction, movement may be 1 byte or 1 bit at a time, providing degrees of speed and smoothness. When scrolling left or right, the basic unit is 1 byte, while the basic unit is 1 bit when scrolling up or down (see details of the screen layout, in Chapter 1). Byte scrolling may be turned into bit scrolling left or right by using other specially designed routines, while byte scrolling may be achieved upwards or downwards by repeating the scrolling action eight times. This slightly reduces the number of routines required.

The terms 'wrap' and 'no wrap' refer to the presence or absence of a built-in wraparound effect in which pixels leaving the screen at one edge re-appear at the opposite edge.

Fast scrolling

Several particularly fast routines are available for scrolling right or left. These routines all act on the display file only. Because of the way in which the display file is constructed, these fast scrolling routines can treat the display in 3 sections – top, centre, and bottom. This means some of the routines can be modified to scroll part of the screen only, which can be very useful.

DISPLAY: FAST BYTE LEFT (NO WRAP)

Scrolls the display left 1 byte without wraparound

LENGTH 22 bytes

Using the routine

No values need be set before executing the routine.

Note: The 3rd data item may be changed to begin scrolling at a different section of the screen. 64 begins at the top of the screen, 72 begins at the centre, and 80 begins at the bottom. The 5th data item controls the number of sections of the screen which are scrolled. 64 scrolls 1 section, 128 two sections, and 192 three sections.

Take care which combination of values is used for the 3rd and 5th data items, since some pairs will be invalid, e.g., 80 and 192 mean start at the bottom section and scroll 3 sections.

This is clearly impossible (sections are not counted backwards).

Display: fast byte left (no wrap) data

33, 0, 64*, 6, 192*, 197, 6, 31, 35, 94,
43, 115, 35, 16, 249, 54, 0, 35, 193, 16,
240, 201

DISPLAY: FAST BYTE LEFT (WRAP)

Scrolls the display left 1 byte with wraparound

LENGTH 22 bytes

Using the routine

No values need be set before executing the routine.

Note: The same modifications to the 3rd and 5th data items are possible, as for DISPLAY: FAST BYTE LEFT (NO WRAP).

Display: fast byte left (wrap) data

33, 0, 64*, 6, 192*, 197, 6, 31, 86, 35,

94, 43, 115, 35, 16, 249, 114, 35, 193, 16,
240, 201

DISPLAY: FAST BYTE RIGHT (NO WRAP)

Scrolls the display right 1 byte without wraparound

LENGTH 23 bytes

Using the routine
No values need be set before executing the routine.

Note: The 3rd data item may be changed to begin scrolling at a different section of the screen. 87 begins at the bottom of the screen, 79 begins at the centre and 71 begins at the top.

The 5th data item controls the number of sections of the screen to be scrolled. 64 scrolls 1 section, 128 scrolls 2 sections, and 192 scrolls all three sections.

Note that the sections are counted from the *bottom* of the screen, so that 87 (3rd item) and 64 (5th item) scrolls only the bottom section of the screen.

Note too, that this routine, in common with the other fast scrolling routines, scrolls the bottom two PRINT lines of the screen (which are normally reserved for input). This usually has no visible effect, unless something has been placed in this section of the screen.

Display: fast byte right (no wrap) data

33, 255, 87*, 6, 192*, 197, 6, 31, 43, 94,
35, 115, 43, 16, 249, 54, 0, 43, 193,
16, 240, 201

DISPLAY: FAST BYTE RIGHT (WRAP)

Scrolls the display right 1 byte with wraparound

LENGTH 22 bytes

Using the routine
No values need be set before executing the routine.

Note: The same modifications to the 3rd and 5th data items are possible, as for DISPLAY: FAST BYTE RIGHT (NO WRAP).

Display: fast byte right (wrap) data

33, 255, 87*, 6, 192*, 197, 6, 31, 86, 43,
94, 35, 115, 43, 16, 249, 114, 43, 193, 16,
240, 201

Example 7.1

Both right and left scrolling routines are used in this example. The wraparound versions of the routines are used, but the 'no wrap' versions may be substituted if required, and used in the same way, although with a rather different result.

```
  10 REM THREE SECTION SCROLLING
  20 REM set colours
  30 PAPER 6: INK 2: BORDER 0
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65301: REM 16K=32533
 120 LET start1=65302: REM 16K=3
2534
 130 LET start2=start1+22
 140 LET start3=start2+22
 150 LET s=start1
 160 LET length=22+22+22
 170 FOR i=1 TO length
 180 READ n
 190 POKE s,n
 200 LET s=s+1
 210 NEXT i
 300 REM print message
 310 PRINT AT 4,14;"THREE"
 320 PRINT AT 12,13;"SECTION"
 330 PRINT AT 20,12;"SCROLLING"
 400 REM scroll sections
 410 FOR i=1 TO 32*10
 420 RANDOMIZE USR start1
 430 RANDOMIZE USR start2
 440 RANDOMIZE USR start3
 450 NEXT i
 500 REM finished
 510 PAPER 7: INK 0: BORDER 7
 520 CLS
1000 REM DISPLAY  FAST BYTE RIGH
T (WRAP)
1010 REM CONFIGURED FOR TOP SECT
```

Program 7.1 *(continues)*

```
ION ONLY
1020 DATA 33,255,71,6,64,197,6,3
1,86,43
1030 DATA 94,35,115,43,16,249,11
4,43,193,16
1040 DATA 240,201
2000 REM DISPLAY  FAST BYTE LEFT
 (WRAP)
2010 REM CONFIGURED FOR CENTRE S
ECTION ONLY
2020 DATA 33,0,72,6,64,197,6,31,
86,35
2030 DATA 94,43,115,35,16,249,11
4,35,193,16
2040 DATA 240,201
3000 REM DISPLAY  FAST BYTE RIGH
T (WRAP)
3010 REM CONFIGURED FOR BOTTOM S
ECTION ONLY
3020 DATA 33,255,87,6,64,197,6,3
1,86,43
3030 DATA 94,35,115,43,16,249,11
4,43,193,16
3040 DATA 240,201
```

Program 7.1

Scrolling 1 byte at a time produces a fast scroll, but the contents of the screen move in rather large increments.

A rather more smooth effect results if scrolling takes place 1 bit (i.e., 1 pixel) at a time. This technique requires 8 times as many scrolling movements to move any item the same distance as the byte scrolling routines, so it is a little slower, but it more than makes up for this with its 'gliding' effect.

Combining the routines for byte and bit scrolling allows an effect similar to acceleration, by using the bit scrolling until this has been executed a sufficient number of times to warrant execution of the byte routine for the same type of scroll. Deceleration employs the reverse technique.

DISPLAY: FAST BIT LEFT (NO WRAP)

Scrolls the display left 1 bit without wraparound

LENGTH 31 bytes

Using the routine

No values need be set before executing the routine.

Note: The same modifications are possible to the 3rd and 5th data items as for the DISPLAY: FAST BYTE LEFT (NO WRAP) routine.

Display: fast bit left (no wrap) data

33, 0, 64*, 6, 192*, 197, 6, 31, 35, 94,
203, 19, 43, 94, 203, 19, 115, 35, 16, 244,
55, 63, 94, 203, 19, 115, 35, 193, 16, 231,
201

DISPLAY: FAST BIT LEFT (WRAP)

Scrolls the display left 1 bit with wraparound

LENGTH 34 bytes

Using the routine

No values need be set before using the routine.

Note: The same modifications may be made to the 3rd and 5th data items as for the DISPLAY: FAST BYTE LEFT (NO WRAP) routine.

Display: fast bit left (wrap) data

33, 0, 64*, 6, 192*, 197, 6, 31, 94, 203,
19, 245, 35, 94, 203, 19, 43, 94, 203, 19,
115, 35, 16, 244, 241, 94, 203, 19, 115, 35,
193, 16, 228, 201

DISPLAY: FAST BIT RIGHT (NO WRAP)

Scrolls the display right 1 bit without wraparound

LENGTH 31 bytes

Using the routine

No values need be set before using the routine.

Note: The same modifications are possible to the 3rd and 5th data items as for the DISPLAY: FAST BYTE RIGHT (NO WRAP) routine.

Display: fast bit right (no wrap) data

33, 255, 87*, 6, 192*, 197, 6, 31, 43, 94,
203, 27, 35, 94, 203, 27, 115, 43, 16, 244,
55, 63, 94, 203, 27, 115, 43, 193, 16, 231,
201

DISPLAY: FAST BIT RIGHT (WRAP)

Scrolls the display right 1 bit with wraparound

LENGTH 34 bytes

Using the routine

No values need be set before executing the routine.

Note: The same modifications may be made to the 3rd and 5th data items as for the DISPLAY: FAST BYTE RIGHT (NO WRAP) routine.

Display: fast bit right (wrap) data

33, 255, 87*, 6, 192*, 197, 6, 31, 94, 203,
27, 245, 43, 94, 203, 27, 35, 94, 203, 27,
115, 43, 16, 244, 241, 94, 203, 27, 115, 43,
193, 16, 228, 201

Example 7.2

Like Example 7.1, this example illustrates the use of two scrolling routines; bit scrolling this time, instead of byte scrolling. As before, the wraparound versions of the routines are used, since these give more opportunity to try the scrolling before the contents of the display file are lost. Try the no wrap versions of the routines, as well as modifying the 3rd and 5th data items in BOTH routines. Use the cursor right and cursor left keys to scroll in the desired direction. A quick dab on one of these keys will scroll the screen 1 bit

at a time, while holding the key down will result in much larger movement caused by repeatedly scrolling 1 bit at a time. Nimble fingers are needed, to scroll just 1 bit!

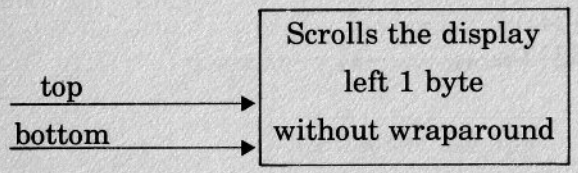
```
  10 REM LUNARSCAPE
  20 REM set colours
  30 PAPER 7: INK 0: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65299: REM 16K=32531
 120 LET start1=65300: REM 16K=3
2532
 130 LET start2=start1+34
 140 LET s=start1
 150 LET length=34+34
 160 FOR i=1 TO length
 170 READ n
 180 POKE s,n
 190 LET s=s+1
 200 NEXT i
 300 REM draw landscape
 310 PLOT 0,10: DRAW 20,30: DRAW
 20,-10: DRAW 10,10: DRAW 10,-20
: DRAW 20,0: DRAW 20,-20: DRAW 5
,10: DRAW 5,-5: DRAW 10,35: DRAW
 10,-20: DRAW 10,0: DRAW 10,10:
DRAW 20,-20: DRAW 20,0: DRAW 30,
-10: DRAW 35,10
 400 REM scroll
 410 LET k$=INKEY$
 420 IF k$=CHR$ 8 THEN  RANDOMIZ
E USR start1
 430 IF k$=CHR$ 9 THEN  RANDOMIZ
E USR start2
 440 IF k$<>"s" THEN  GO TO 410
 500 REM finished
1000 REM DISPLAY  FAST BIT LEFT
(WRAP)
1010 DATA 33,0,80,6,64,197,6,31,
94,203
1020 DATA 19,245,35,94,203,19,43
,94,203,19
1030 DATA 115,35,16,244,241,94,2
03,19,115,35
1040 DATA 193,16,228,201
2000 REM DISPLAY  FAST BIT RIGHT
 (WRAP)
2010 DATA 33,255,87,6,64,197,6,3
1,94,203
2020 DATA 27,245,43,94,203,27,35
,94,203,27
2030 DATA 115,43,16,244,241,94,2
03,27,115,43
2040 DATA 193,16,228,201
```

Program 7.2

Programmable scrolling

The next set of routines are a little longer, and execute a little more slowly. However, they do allow scrolling between any two horizontal lines on the display. The loss of speed is slight, and makes little difference in practice, but the ability to scroll between defined limits (instead of by sections) provides a great deal of extra flexibility.

DISPLAY: BYTE LEFT (NO WRAP)



LENGTH 94 bytes

Using the routine

POKE the *y* co-ordinate of the top of the scrolling section into location 23296.
POKE the *y* co-ordinate of the bottom of the scrolling section into location 23297.

Note: Returns to BASIC, without scrolling, if the top of the scrolling area is lower than, or equal to, the bottom; or if the top or bottom are greater than 175.

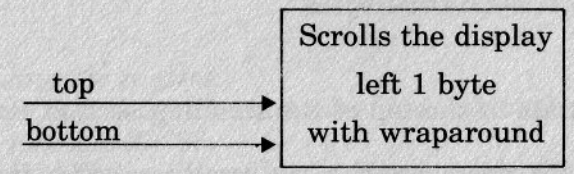
Display: byte left (no wrap) data

```
58, 1, 91, 254, 176, 48, 86, 87, 58, 0,

91, 254, 176, 48, 78, 186, 56, 75, 40, 73,

213, 245, 24, 19, 241, 209, 6, 31, 35, 94,

43, 115, 35, 16, 249, 54, 0, 186, 40, 53,

61, 24, 233, 71, 62, 16, 128, 71, 33, 0,

64, 254, 128, 48, 9, 17, 0, 8, 25, 254,

64, 48, 1, 25, 203, 184, 203, 176, 62, 63,
```



```
144, 17, 32, 0, 254, 8, 56, 5, 214, 8,

25, 24, 247, 254, 0, 40, 4, 61, 36, 24,

248, 24, 187, 201
```

DISPLAY: BYTE LEFT (WRAP)



LENGTH 94 bytes

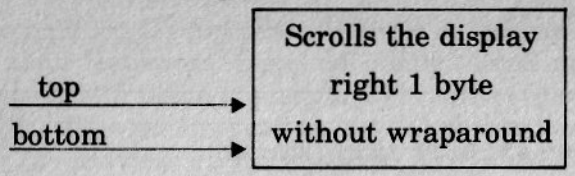
Using the routine

POKE the *y* co-ordinate of the top of the scrolling section into location 23296
POKE the *y* co-ordinate of the bottom of the scrolling section into location 23297.

Note: Incorporates the same safety checks as DISPLAY: BYTE LEFT (NO WRAP).

Display: byte left (wrap) data

```
58, 1, 91, 254, 176, 48, 86, 87, 58, 0,

91, 254, 176, 48, 78, 186, 56, 75, 40, 73,

213, 245, 24, 19, 241, 209, 78, 6, 31, 35,

94, 43, 115, 35, 16, 249, 113, 186, 40, 53,

61, 24, 233, 71, 62, 16, 128, 71, 33, 0,

64, 254, 128, 48, 9, 17, 0, 8, 25, 254,

64, 48, 1, 25, 203, 184, 203, 176, 62, 63,

144, 17, 32, 0, 254, 8, 56, 5, 214, 8,

25, 24, 247, 254, 0, 40, 4, 61, 36, 24,

248, 24, 187, 201
```



LENGTH 94 bytes

Using the routine

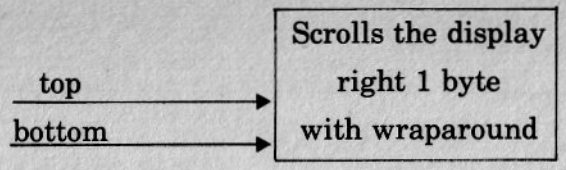
POKE the *y* co-ordinate of the top of the scrolling section into location 23296.

POKE the *y* co-ordinate of the bottom of the scrolling section into location 23297.

Note: Incorporates the same safety checks as DISPLAY: BYTE LEFT (NO WRAP).

Display: byte right (no wrap) data

```
58, 1, 91, 254, 176, 48, 86, 87, 58, 0,

91, 254, 176, 48, 78, 186, 56, 75, 40, 73,

213, 245, 24, 19, 241, 209, 6, 31, 43, 94,

35, 115, 43, 16, 249, 54, 0, 186, 40, 53,

61, 24, 233, 71, 62, 16, 128, 71, 33, 31,

64, 254, 128, 48, 9, 17, 0, 8, 25, 254,

64, 48, 1, 25, 203, 184, 203, 176, 62, 63,

144, 17, 32, 0, 254, 8, 56, 5, 214, 8,

25, 24, 247, 254, 0, 40, 4, 61, 36, 24,

248, 24, 187, 201
```

LENGTH 94 bytes

Using the routine

POKE the *y* co-ordinate of the top of the scrolling section into location 23296.

POKE the *y* co-ordinate of the bottom of the scrolling section into location 23297.

Note: Incorporates the same safety checks as DISPLAY: BYTE LEFT (NO WRAP).

Display: byte right (wrap) data

```
58, 1, 91, 254, 176, 48, 86, 87, 58, 0,

91, 254, 176, 48, 78, 186, 56, 75, 40, 73,

213, 245, 24, 19, 241, 209, 78, 6, 31, 43,

94, 35, 115, 43, 16, 249, 113, 186, 40, 53,

61, 24, 233, 71, 62, 16, 128, 71, 33, 31,

64, 254, 128, 48, 9, 17, 0, 8, 25, 254,

64, 48, 1, 25, 203, 184, 203, 176, 62, 63,

144, 17, 32, 0, 254, 8, 56, 5, 214, 8,

25, 24, 247, 254, 0, 40, 4, 61, 36, 24,

248, 24, 187, 201
```

Example 7.3

```
10 REM TRAFFIC
20 REM set colours
30 PAPER 0: INK 7: BORDER 7
```

Program 7.3 *(continues)*

```
 40 CLS
 100 REM set new ramtop
 110 CLEAR 65179: REM 16K=32411
 120 LET start1=65180: REM 16K=3
2412
 130 LET start2=start1+94
 140 LET s=start1
 150 LET length=94+94
 160 FOR i=1 TO length
 170 READ n
 180 POKE s,n
 190 LET s=s+1
 200 NEXT i
 250 REM define variables for sc
rolling sections
 260 LET t=23296
 270 LET b=23297
 300 REM define UDG characters
 310 FOR i=1 TO 8
 320 READ c$
 330 FOR j=0 TO 7
 340 READ d
 350 POKE USR c$+j,d
 360 NEXT j
 370 NEXT i
 400 REM initialise streams of t
raffic
 410 LET a$=CHR$ 32+CHR$ 144+CHR
$ 145+CHR$ 32
 420 LET b$=CHR$ 146+CHR$ 147+CH
R$ 32+CHR$ 32
 430 LET c$=CHR$ 32+CHR$ 148+CHR
$ 149+CHR$ 32
 440 LET d$=CHR$ 150+CHR$ 151+CH
R$ 32+CHR$ 32
 450 FOR i=1 TO 3
 460 LET a$=a$+a$
 470 LET b$=b$+b$
 480 LET c$=c$+c$
 490 LET d$=d$+d$
 500 NEXT i
 600 REM draw traffic
 610 PRINT  INK 6;AT 1,0;b$
 620 PRINT  INK 5;AT 3,0;c$
 630 PRINT AT 4,0;d$
 640 PRINT  INK 4;AT 6,0;a$
 650 PRINT AT 7,0;b$
 660 PRINT  INK 6;AT 9,0;c$
 670 PRINT  INK 5;AT 11,0;b$
 680 PRINT AT 13,0;c$
 690 PRINT  INK 6;AT 14,0;d$
 700 PRINT AT 16,0;a$
 710 PRINT  INK 4;AT 17,0;b$
 720 PRINT AT 19,0;c$
 800 REM scroll
```

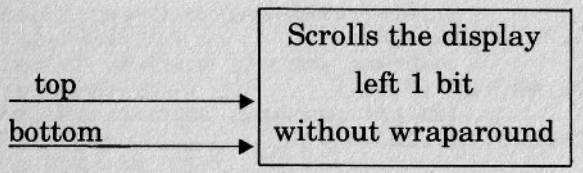
Program 7.3 *(continues)*


```
 810 FOR i=1 TO 100
 820 POKE t,167: POKE b,160: RAN
DOMIZE USR start1
 830 POKE t,151: POKE b,136: RAN
DOMIZE USR start2
 840 POKE t,127: POKE b,112: RAN
DOMIZE USR start1
 850 POKE t,103: POKE b,96: RAND
OMIZE USR start2
 860 POKE t,87: POKE b,80: RANDO
MIZE USR start1
 870 POKE t,71: POKE b,56: RANDO
MIZE USR start2
 880 POKE t,47: POKE b,32: RANDO
MIZE USR start1
 890 POKE t,23: POKE b,16: RANDO
MIZE USR start2
 900 NEXT i
 950 REM finished
 960 INK 0: PAPER 7
1000 REM DISPLAY  BYTE RIGHT (WR
AP)
1010 DATA 58,1,91,254,176,48,86,
87,58,0
1020 DATA 91,254,176,48,78,186,5
6,75,40,73
1030 DATA 213,245,24,19,241,209,
78,6,31,43
1040 DATA 94,35,115,43,16,249,11
3,186,40,53
1050 DATA 61,24,233,71,62,16,128
,71,33,31
1060 DATA 64,254,128,48,9,17,0,8
,25,254
1070 DATA 64,48,1,25,203,184,203
,176,62,63
1080 DATA 144,17,32,0,254,8,56,5
,214,8
1090 DATA 25,24,247,254,0,40,4,6
1,36,24
1100 DATA 248,24,187,201
2000 REM DISPLAY  BYTE LEFT (WRA
P)
2010 DATA 58,1,91,254,176,48,86,
87,58,0
2020 DATA 91,254,176,48,78,186,5
6,75,40,73
2030 DATA 213,245,24,19,241,209,
78,6,31,35
2040 DATA 94,43,115,35,16,249,11
3,186,40,53
2050 DATA 61,24,233,71,62,16,128
,71,33,0
2060 DATA 64,254,128,48,9,17,0,8
,25,254
```

```
2070 DATA 64,48,1,25,203,184,203
,176,62,63
2080 DATA 144,17,32,0,254,8,56,5
,214,8
2090 DATA 25,24,247,254,0,40,4,6
1,36,24
2100 DATA 248,24,187,201
3000 REM UDG DEFINITIONS
3010 DATA "A",0,31,34,66,255,255
,127,48
3020 DATA "B",0,128,64,32,252,25
4,255,24
3030 DATA "C",127,127,127,127,12
7,127,127,48
3040 DATA "D",240,240,144,144,25
4,254,254,48
3050 DATA "E",0,1,2,4,63,127,255
,24
3060 DATA "F",0,248,68,66,255,25
5,254,12
3070 DATA "G",15,15,9,9,127,127,
127,24
3080 DATA "H",254,254,254,254,25
4,254,254,12
```

Program 7.3

DISPLAY: BIT LEFT (NO WRAP)



LENGTH 105 bytes

Using the routine

POKE the *y* co-ordinate of the top of the scrolling section into location 23296.

POKE the *y* co-ordinate of the bottom of the scrolling section into location 23297.

Note: Incorporates the same safety checks as DISPLAY: BYTE LEFT (NO WRAP).

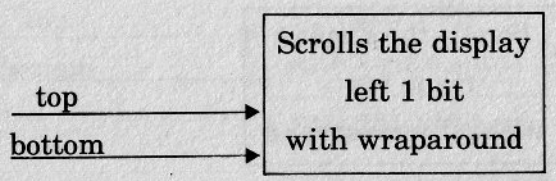
Display: bit left (no wrap) data

```
58, 1, 91, 254, 176, 48, 97, 87, 58, 0,

91, 254, 176, 48, 89, 186, 56, 86, 40, 84,
```



```
213, 245, 24, 30, 241, 209, 6, 31, 245, 35,

94, 203, 19, 43, 94, 203, 19, 115, 35, 55,

63, 16, 242, 94, 203, 19, 115, 241, 186, 40,

53, 61, 24, 222, 71, 62, 16, 71, 128, 71, 33,

0, 64, 254, 128, 48, 9, 17, 0, 8, 25,

254, 64, 48, 1, 25, 203, 184, 203, 176, 62,

63, 144, 17, 32, 0, 254, 8, 56, 5, 214,

8, 25, 24, 247, 254, 0, 40, 4, 61, 36,

24, 248, 24, 176, 201
```

DISPLAY: BIT LEFT (WRAP)



LENGTH 112 bytes

Using the routine

POKE the *y* co-ordinate of the top of the scrolling section into location 23296.

POKE the *y* co-ordinate of the bottom of the scrolling section into location 23297.

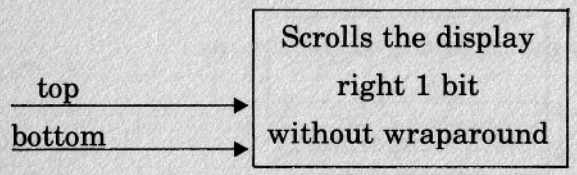
Note: Incorporates the same safety checks as DISPLAY: BYTE LEFT (NO WRAP).

Display: bit left (wrap) data

```
58, 1, 91, 254, 176, 48, 104, 87, 58, 0,

91, 254, 176, 48, 96, 186, 56, 93, 40, 91,

213, 245, 24, 37, 241, 209, 6, 31, 245, 94,
```

```
203, 19, 245, 55, 63, 35, 94, 203, 19, 43,
94, 203, 19, 115, 35, 55, 63, 16, 242, 241,
94, 203, 19, 115, 241, 186, 40, 53, 61, 24
215, 71, 62, 16, 128, 71, 33, 0, 64, 254,
128, 48, 9, 17, 0, 8, 25, 254, 64, 48,
1, 25, 203, 184, 203, 176, 62, 63, 144,17,
32, 0, 254, 8, 56, 5, 214, 8, 25, 24,
247, 254, 0, 40, 4, 61, 36, 24, 248, 24,
169, 201
```

DISPLAY: BIT RIGHT (NO WRAP)



LENGTH 105 bytes

Using the routine

POKE the *y* co-ordinate of the top of the scrolling section into location 23296.
POKE the *y* co-ordinate of the bottom of the scrolling section into location 23297.

Note: Incorporates the same safety checks as DISPLAY: BYTE LEFT (NO WRAP).

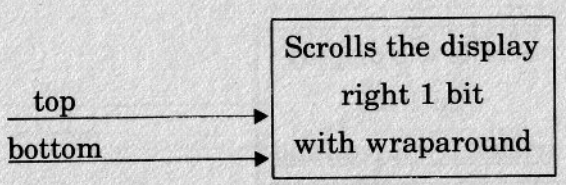
Display: bit right (no wrap) data

```
58, 1, 91, 254, 176, 48, 97, 87, 58, 0,
91, 254, 176, 48, 89, 186, 56, 86, 40, 84,
213, 245, 24, 30, 241, 209, 6, 31, 245, 43,
```



```
94, 203, 27, 35, 94, 203, 27, 115, 43, 55,
63, 16, 242, 94, 203, 27, 115, 241, 186, 40,
53, 61, 24, 222, 71, 62, 16, 128, 71, 33,
31, 64, 254, 128, 48, 9, 17, 0, 8, 25,
254, 64, 48, 1, 25, 203, 184, 203, 176, 62,
63, 144, 17, 32, 0, 254, 8, 56, 5, 214,
8, 25, 24, 247, 254, 0, 40, 4, 61, 36,
24, 248, 24, 176, 201
```

DISPLAY: BIT RIGHT (WRAP)



LENGTH 112 bytes

Using the routine

POKE the *y* co-ordinate of the top of the scrolling section into location 23296.
POKE the *y* co-ordinate of the bottom of the scrolling section into location 23297.

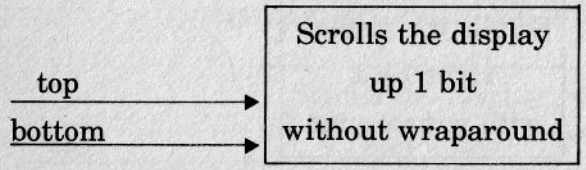
Note: Incorporates the same safety checks as DISPLAY: BYTE LEFT (NO WRAP).

Display: bit right (wrap) data

```
58, 1, 91, 254, 176, 48, 104, 87, 58, 0,
91, 254, 176, 48, 96, 186, 56, 93, 40, 91,
213, 245, 24, 37, 241, 209, 6, 31, 245, 94,
203, 27, 245, 55, 63, 43, 94, 203, 27, 35,
```

```
94, 203, 27, 115, 43, 55, 63, 16, 242, 241,
94, 203, 27, 115, 241, 186, 40, 53, 61, 24,
215, 71, 62, 16, 128, 71, 33, 31, 64, 254,
128, 48, 9, 17, 0, 8, 25, 254, 64, 48,
1, 25, 203, 184, 203, 176, 62, 63, 144, 17,
32, 0, 254, 8, 56, 5, 214, 8, 25, 24,
247, 254, 0, 40, 4, 61, 36, 24, 248, 24,
169, 201
```

DISPLAY: BIT UP (NO WRAP)



LENGTH 118 bytes

Using the routine

POKE the *y* co-ordinate of the top of the scrolling section into location 23296.
POKE the *y* co-ordinate of the bottom of the scrolling section into location 23297.

Note: Incorporates the same safety checks as DISPLAY: BYTE LEFT (NO WRAP).

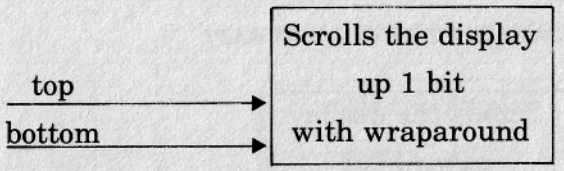
Display: bit up (no wrap) data

```
58, 1, 91, 254, 176, 48, 110, 71, 58, 0,
91, 254, 176, 48, 102, 184, 56, 99, 40, 97,
197, 14, 255, 245, 24, 36, 241, 14, 0, 61,
245, 229, 24, 28, 209, 213, 229, 1, 32, 0,
```



```
237, 176, 209, 225, 241, 193, 184, 197, 235, 32,
232, 193, 62, 0, 6, 32, 119, 35, 16, 252,
24, 55, 71, 62, 16, 128, 71, 33, 0, 64,
254, 128, 48, 9, 17, 0, 8, 25, 254, 64,
48, 1, 25, 203, 184, 203, 176, 62, 63, 144,
17, 32, 0, 254, 8, 56, 5, 214, 8, 25,
24, 247, 254, 0, 40, 4, 61, 36, 24, 248,
121, 254, 255, 40, 167, 32, 173, 201
```

DISPLAY: BIT UP (WRAP)



LENGTH 129 bytes

Using the routine

POKE the *y* co-ordinate of the top of the scrolling section into location 23296.
POKE the *y* co-ordinate of the bottom of the scrolling section into location 23297.

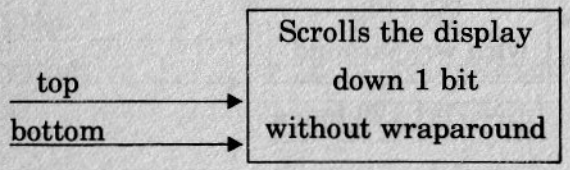
Note: This routine uses locations 23300–23331 inclusive, as workspace. It also incorporates the same safety checks as DISPLAY: BYTE LEFT (NO WRAP)

Display: bit up (wrap) data

```
58, 1, 91, 254, 176, 48, 121, 71, 58, 0,
91, 254, 176, 48, 113, 184, 56, 110, 40, 108,
197, 14, 255, 245, 24, 47, 229, 1, 32, 0,
17, 4, 91, 237, 176, 225, 241, 14, 0, 61,
```

```
245, 229, 24, 29, 209, 213, 229, 1, 32, 0,
237, 176, 209, 225, 241, 193, 184, 197, 235, 32,
232, 193, 1, 32, 0, 235, 33, 4, 91, 237,
176, 24, 5,, 71, 62, 16, 128, 71, 33, 0,
64, 254, 128, 48, 9, 17, 0, 8, 25, 254,
64, 48, 1, 25, 203, 184, 203, 176, 62, 63,
144, 17, 32, 0, 254, 8, 56, 5, 214, 8,
25, 24, 247, 254, 0, 40, 4, 61, 36, 24,
248, 121, 254, 255, 40, 156, 32, 172, 201
```

DISPLAY: BIT DOWN (NO WRAP)



LENGTH 118 bytes

Using the routine

POKE the *y* co-ordinate of the top of the scrolling section into location 23296.
POKE the *y* co-ordinate of the bottom of the scrolling section into location 23297.

Note: Incorporates the same safety checks as DISPLAY: BYTE LEFT (NO WRAP).

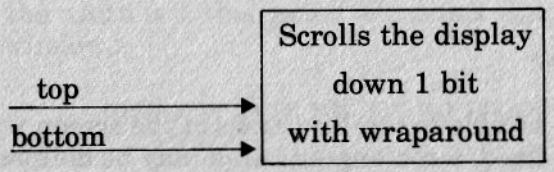
Display: bit down (no wrap) data

```
58, 0, 91, 254, 176, 48, 53, 71, 58, 1,
91, 254, 176, 48, 45, 184, 56, 42, 40, 40,
197, 14, 255, 245, 24, 36, 241, 14, 0, 60,
```



```
245, 229, 24, 28, 209, 213, 229, 1, 32, 0,
237, 176, 209, 225, 241, 193, 184, 197, 235, 32,
232, 193, 62, 0, 6, 32, 119, 35, 16, 252,
24, 55, 71, 62, 16, 128, 71, 33, 0, 64,
254, 128, 48, 9, 17, 0, 8, 25, 254, 64,
48, 1, 25, 203, 184, 203, 176, 62, 63, 144,
17, 32, 0, 254, 8, 56, 5, 214, 8, 25,
24, 247, 254, 0, 40, 4, 61, 36, 24, 248,
121, 254, 255, 40, 167, 32, 173, 201
```

DISPLAY: BIT DOWN (WRAP)



LENGTH 129 bytes

Using the routine

POKE the *y* co-ordinate of the top of the scrolling section into location 23296.
POKE the *y* co-ordinate of the bottom of the scrolling section into location 23297.

Note: This routine uses locations 23300–23331 inclusive, as workspace. It also incorporates the same safety checks as DISPLAY: BYTE LEFT (NO WRAP).

Display: bit down (wrap) data

```
58, 0, 91, 254, 176, 48, 121, 71, 58, 1,
91, 254, 176, 48, 113, 184, 48, 110, 40, 108,
197, 14, 255, 245, 24, 47, 229, 1, 32, 0,
```

```
17, 4, 91, 237, 176, 225, 241, 14, 0, 60,

245, 229, 24, 29, 209, 213, 229, 1, 32, 0,

237, 176, 209, 225, 241, 193, 184,197, 235, 32,

232, 193, 1, 32, 0, 235, 33, 4, 91, 237,

176, 24, 55, 71, 62, 16, 128, 71, 33, 0,

64, 254, 128, 48, 9, 17, 0, 8, 25, 254,

64, 48, 1, 25, 203, 184, 203, 176, 62, 63,

144, 17, 32, 0, 254, 8, 56, 5, 214, 8,

25, 24, 247, 254, 0, 40, 4, 61, 36, 24,

248, 121, 254, 255, 40, 156, 32, 172, 201
```

A scrolling window

Because it is handy to be able to scroll sections of the screen while leaving other sections fixed, a rectangular area may be defined on the screen to set limits on the scrolling. Such an area is called a window. Inside this window, any of the methods of scrolling may be used – byte or bit scrolling; up, down, left or right. All the scrolling routines to be described offer the same form of protection against incorrect initial values. If any value which describes the window is found to be unsuitable, the scrolling does not take place. The routine simply returns to BASIC.

A typical window is shown in Fig. 7.1, and is described by four values:

The *y* co-ordinate of the topmost line of the window.
The *y* co-ordinate of the bottom line in the window.
The left edge of the window.
The width of the window.

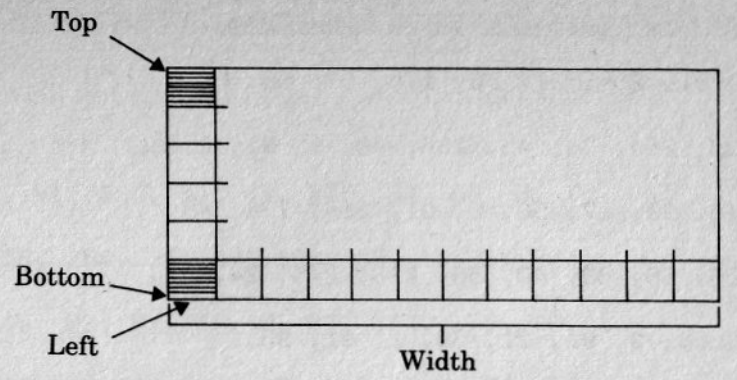
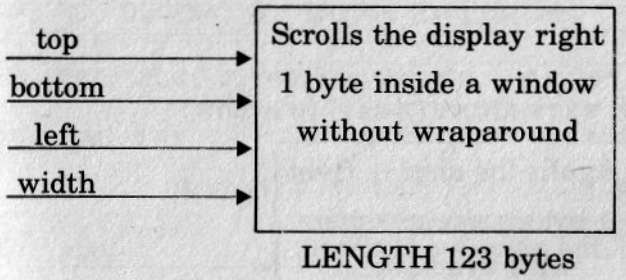


Figure 7.1 A scrolling window

The left edge and the width are always measured using PRINT positions.

Since the basic unit in the vertical direction is 1 bit, the top and bottom of the window are always specified using *y* co-ordinates. The width of the window is the number of PRINT positions inside the window: for example, if the left edge of the window is at column 2, and the width is 5, then print columns 2, 3, 4, 5, and 6 are all inside the window.

DISPLAY: BYTE RIGHT (NO WRAP) (WINDOW)



LENGTH 123 bytes

Using the routine

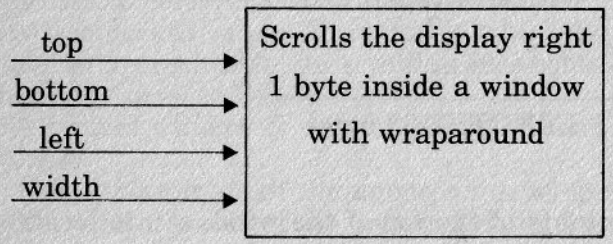
POKE the *y* co-ordinate of the top of the window into location 23296.
POKE the *y* co-ordinate of the bottom of the window into location 23297.
POKE the leftmost PRINT position into location 23298.
POKE the width (number of PRINT positions) into location 23299.

Note: Will not scroll if inappropriate values are used.

Display: byte right (no wrap) (window) data

```
58, 2, 91, 254, 31, 48, 115, 87, 58, 3,
91, 138, 254, 34, 48, 106, 58, 1, 91, 254,
176, 48, 99, 87, 58, 0, 91, 254, 176, 48,
91, 186, 56, 88, 40, 86, 213, 245, 24, 32,
22, 0, 58, 3, 91, 71, 58, 2, 91, 95,
25, 88, 25, 43, 5, 241, 209, 43, 94, 35,
115, 43, 16, 249, 54, 0, 186, 40, 53, 61,
24, 220, 71, 62, 16, 128, 71, 33, 0, 64,
254, 128, 48, 9, 17, 0, 8, 25, 254, 64,
48, 1, 25, 203, 184, 203, 176, 62, 63, 144,
17, 32, 0, 254, 8, 56, 5, 214, 8, 25,
24, 247, 254, 0, 40, 4, 61, 36, 24, 248,
24, 174, 201
```

DISPLAY: BYTE RIGHT (WRAP) (WINDOW)



LENGTH 123 bytes

Using the routine

POKE the *y* co-ordinate of the top of the window into location 23296.

POKE the *y* co-ordinate of the bottom of the window into location 23297.

POKE the leftmost PRINT position into location 23298.

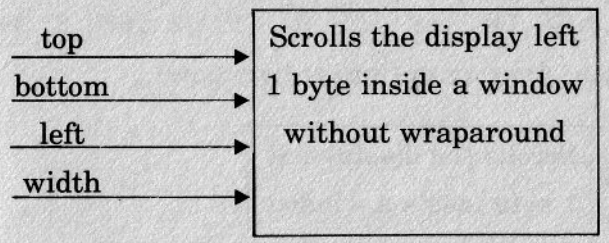
POKE the width (number of PRINT positions) into location 23299.

Note: Will not scroll if inappropriate values are used.

Display: byte right (wrap) (window) data

```
58, 2, 91, 254, 31, 48, 115, 87, 58, 3,
91, 138, 254, 34, 48, 106, 58, 1, 91, 254,
176, 48, 99, 87, 58, 0, 91, 254, 176, 48,
91, 186, 56, 88, 40, 86, 213, 245, 24, 32,
22, 0, 58, 3, 91, 71, 58, 2, 91, 95,
25, 88, 25, 43, 5, 78, 241, 209, 43, 94,
35, 115, 43, 16, 249, 113, 186, 40, 53, 61,
24, 220, 71, 62, 16, 128, 71, 33, 0, 64,
254, 128, 48, 9, 17, 0, 8, 25, 254, 64,
48, 1, 25, 203, 184, 203, 176, 62, 63, 144,
17, 32, 0, 254, 8, 56, 5, 214, 8, 25,
24, 247, 254, 0, 40, 4, 61, 36, 24, 248,
24, 174, 201
```

DISPLAY: BYTE LEFT (NO WRAP) (WINDOW)



LENGTH 120 bytes

Using the routine

POKE the *y* co-ordinate of the top of the window into location 23296.

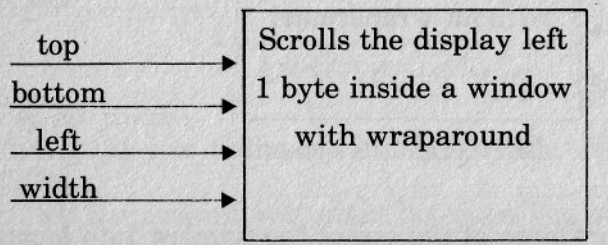
POKE the *y* co-ordinate of the bottom of the window into location 23297.
POKE the leftmost PRINT position into location 23298.
POKE the width (number of PRINT positions) into location 23299.

Note: Will not scroll if inappropriate values are used.

Display: byte left (no wrap) (window) data

```
58, 2, 91, 254, 31, 48, 112, 87, 58, 3,

91, 138, 254, 34, 48, 103, 58, 1, 91, 254,

176, 48, 96, 87, 58, 0, 91, 254, 176, 48,

88, 186, 56, 85, 40, 83, 213, 245, 24, 29,

22, 0, 58, 3, 91, 71, 58, 2, 91, 95,

25, 5, 241, 209, 35, 94, 43, 115, 35, 16,

249, 54, 0, 186, 40, 53, 61, 24, 223, 71,

62, 16, 128, 71, 33, 0, 64, 254, 128, 48,

9, 17, 0, 8, 25, 254, 64, 48, 1, 25,

203, 184, 203, 176, 62, 63, 144, 17, 32, 0,

254, 8, 56, 5, 214, 8, 25, 24, 247, 254,

0, 40, 4, 61, 36, 24, 248, 24, 177, 201
```

DISPLAY: BYTE LEFT (WRAP) (WINDOW)



LENGTH 120 bytes

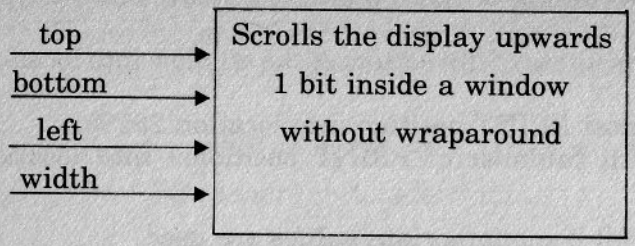
Using the routine
POKE the *y* co-ordinate of the top of the window into location 23296.
POKE the *y* co-ordinate of the bottom of the window into location 23297.
POKE the leftmost PRINT position into location 23298.
POKE the width (number of PRINT positions) into location 23299.

Note: Will not scroll if inappropriate values are used.

Display: byte left (wrap) (window) data

```
58, 2, 91, 254, 31, 48, 112, 87, 58, 3,

91, 138, 254, 34, 48, 103, 58, 1, 91, 254,

176, 48, 96, 87, 58, 0, 91, 254, 176, 48,

88, 186, 56, 85, 40, 83, 213, 245, 24, 29,

22, 0, 58, 3, 91, 71, 58, 2, 91, 95,

25, 5, 241, 209, 78, 35, 94, 43, 115, 35,

16, 249, 113, 186, 40, 53, 61, 24, 223, 71,

62, 16, 128, 71, 33, 0, 64, 254, 128, 48,

9, 17, 0, 8, 25, 254, 64, 48, 1, 25,

203, 184, 203, 176, 62, 63, 144, 17, 32, 0,

254, 8, 56, 5, 214, 8, 25, 24, 247, 254,

0, 40, 4, 61, 36, 24, 248, 24, 177, 201
```

DISPLAY: BIT UP (NO WRAP) (WINDOW)



LENGTH 146 bytes

Using the routine

POKE the *y* co-ordinate of the top of the window into location 23296.
POKE the *y* co-ordinate of the bottom of the window into location 23297.
POKE the leftmost PRINT position into location 23298.
POKE the width (number of PRINT positions) into location 23299.

Note: Will not scroll if inappropriate values are used.

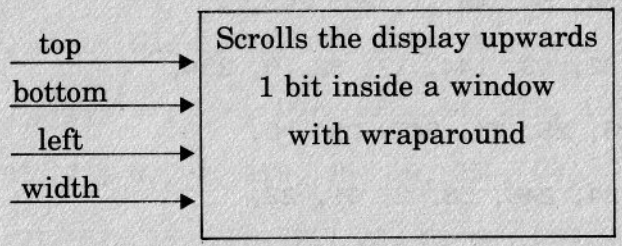
Display: bit up (no wrap) (window) data

```
58, 2, 91, 254, 31, 48, 138, 87, 58, 3,

91, 138, 254, 34, 48, 129, 58, 1, 91, 254,

176, 48, 122, 71, 58, 0, 91, 254, 176, 48,

114, 184, 56, 111, 40, 109, 197, 14, 255, 245,

24, 41, 241, 14, 0, 61, 245, 229, 24, 33,

209, 213, 229, 6, 0, 58, 3, 91, 79, 237,

176, 209, 225, 241, 193, 184, 197, 235, 32, 229,

193, 58, 3, 91, 71, 62, 0, 119, 35, 16,

252, 24, 62, 71, 62, 16, 128, 71, 33, 0,

64, 254, 128, 48, 9, 17, 0, 8, 25, 254,
```



```
64, 48, 1, 25, 203, 184, 203, 176, 62, 63,

144, 17, 32, 0, 254, 8, 56, 5, 214, 8,

25, 24, 247, 254, 0, 40, 4, 61, 36, 24,

248, 58, 2, 91, 22, 0, 95, 25, 121, 254,

255, 40, 155, 32, 161, 201
```

DISPLAY: BIT UP (WRAP) (WINDOW)



LENGTH 161 bytes

Using the routine

POKE the *y* co-ordinate of the top of the window into location 23296.
POKE the *y* co-ordinate of the bottom of the window into location 23297.
POKE the leftmost PRINT position into location 23298.
POKE the width (number of PRINT positions) into location 23299.

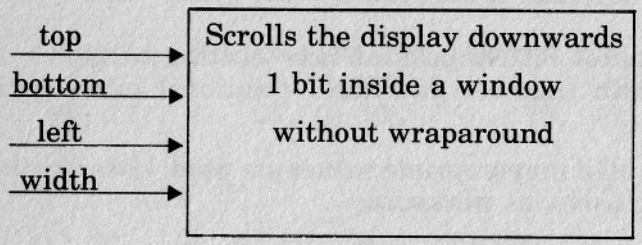
Note: Will not scroll if inappropriate values are used. Uses locations 23300–23331 inclusive, as workspace.

Display: bit up (wrap) (window) data

```
58, 2, 91, 254, 31, 48, 153, 87, 58, 3,

91, 138, 254, 34, 48, 144, 58, 1, 91, 254,

176, 48, 137, 71, 58, 0, 91, 254, 176, 48,

129, 184, 56, 126, 40, 124, 197, 14, 255, 245,

24, 56, 229, 6, 0, 58, 3, 91, 79, 17,
```

```
4, 91, 237, 176, 225, 241, 14, 0, 61, 245,
229, 24, 35, 209, 213, 229, 6, 0, 58, 3,
91, 79, 237, 176, 209, 225, 241, 193, 184, 197,
235, 32, 229, 193, 58, 3, 91, 79, 6, 0,
235, 33, 4, 91, 237, 176, 24, 62, 71, 62,
16, 128, 71,33, 0, 64, 254, 128, 48, 9,
17, 0, 8, 25, 254, 64, 48, 1, 25, 203,
184, 203, 176, 62, 63, 144, 17, 32, 0, 254,
8, 56, 5, 214, 8, 25, 24, 247, 254, 0,
40, 4, 61, 36, 24, 248, 58, 2, 91, 22,
0, 95, 25, 121, 254, 255, 40, 140, 32, 159,
201
```

DISPLAY: BIT DOWN (NO WRAP) (WINDOW)



LENGTH 144 bytes

Using the routine

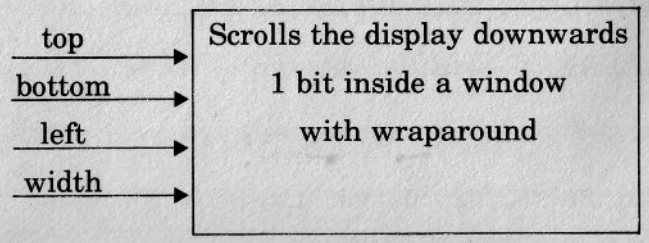
POKE the *y* co-ordinate of the top of the window into location 23296.
POKE the *y* co-ordinate of the bottom of the window into location 23297.
POKE the leftmost PRINT position into location 23298.
POKE the width (number of PRINT positions) into location 23299.

Note: Will not scroll if inappropriate values are used.

Display: bit down (no wrap) (window) data

```
58, 2, 91, 254, 31, 48, 136, 87, 58, 3,
91, 138, 254, 34, 48, 127, 58, 0, 91, 254,
176, 48, 120, 71, 58, 1, 91, 254, 176, 48,
112, 184, 48, 109, 197, 14, 255, 245, 24, 41,
241, 14, 0, 60, 245, 229, 24, 33, 209, 213,
229, 6, 0, 58, 3, 91, 79, 237, 176, 209,
225, 241, 193, 184, 197, 235, 32, 229, 193, 58,
3, 91, 71, 62, 0, 119, 35, 16, 252, 24,
62, 71, 62, 16, 128, 71, 33, 0, 64, 254,
128, 48, 9, 17, 0, 8, 25, 254, 64, 48,
1, 25, 203, 184, 203, 176, 62, 63, 144, 17,
32, 0, 254, 8, 56, 5, 214, 8, 25, 24,
247, 254, 0, 40, 4, 61, 36, 24, 248, 58,
2, 91, 22, 0, 95, 25, 121, 254, 255, 40,
155, 32, 161, 201
```

DISPLAY: BIT DOWN (WRAP) (WINDOW)



LENGTH 158 bytes

Using the routine

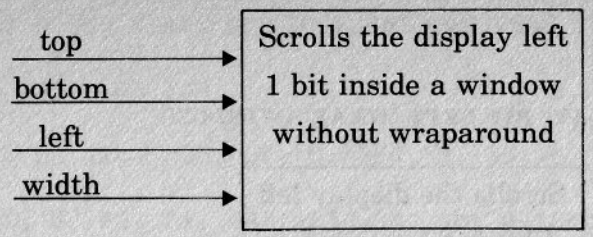
POKE the *y* co-ordinate of the top of the window into location 23296.
POKE the *y* co-ordinate of the bottom of the window into location 23297.
POKE the leftmost PRINT position into location 23298.
POKE the width (number of PRINT positions) into location 23299.

Note: Will not scroll if inappropriate values are used. Uses locations 23300–23331 inclusive as workspace.

Display: bit down (wrap) (window) data

```
58, 2, 91, 254, 31, 48, 150, 87, 58, 3,

91, 138, 254, 34, 48, 141, 58, 0, 91, 254,

176, 48, 134, 71, 58, 1, 91, 254, 176, 48,

126, 184, 48, 123, 197, 14, 255, 245, 24, 55,

229, 6, 0, 58, 3, 91, 79, 17, 4, 91,

237, 176, 225, 241, 14, 0, 60, 245, 229, 24,

34, 209, 213, 229, 6, 0, 58, 3, 91, 79,

237, 176, 225, 209, 241, 193, 184, 197, 32, 230,

193, 58, 3, 91, 79, 6, 0, 235, 33, 4,

91, 237, 176, 24, 62, 71, 62, 16, 128, 71,

33, 0, 64, 254, 128, 48, 9, 17, 0, 8,

25, 254, 64, 48, 1, 25, 203, 184, 203, 176,

62, 63, 144, 17, 32, 0, 254, 8, 56, 5,

214, 8, 25, 24, 247, 254, 0, 40, 4, 61,

36, 24, 248, 58, 2, 91, 22, 0, 95, 25,

121, 254, 255, 40, 141, 32, 160, 201
```


DISPLAY: BIT LEFT (NO WRAP) (WINDOW)



LENGTH 129 bytes

Using the routine

POKE the *y* co-ordinate of the top of the window into location 23296.
POKE the *y* co-ordinate of the bottom of the window into location 23297.
POKE the leftmost PRINT position into location 23298.
POKE the width (number of PRINT positions) into location 23299.

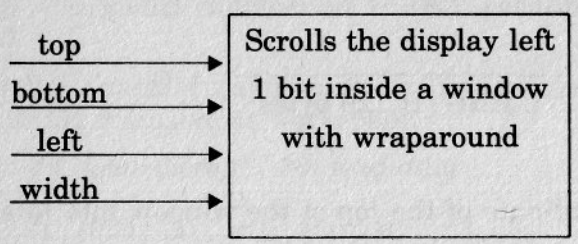
Note: Will not scroll if inappropriate values are used.

Display: bit left (no wrap) (window) data

```
58, 2, 91, 254, 176, 48, 121, 87, 58, 3,

91, 138, 254, 34, 48, 112, 58, 1, 91, 254,

176, 48, 105, 87, 58, 0, 91, 254, 176, 48,

97, 186, 56, 94, 40, 92, 213, 245, 24, 38,

22, 0, 58, 3, 91, 71, 58, 2, 91, 95,

25, 5, 35, 94, 203, 19, 43, 94, 203, 19,

115, 35, 55, 63, 16, 242, 94, 203, 19, 115,

241, 209, 186, 40, 53, 61, 24, 214, 71, 62,

16, 128, 71, 33, 0, 64, 254, 128, 48, 9,

17, 0, 8, 25, 254, 64, 48, 1, 25, 203,

184, 203, 176, 62, 63, 144, 17, 32, 0, 254,
```

```
8, 56, 5, 214, 8, 25, 24, 247, 254, 0,

40, 4, 61, 36, 24, 248, 24, 168, 201
```

DISPLAY: BIT LEFT (WRAP) (WINDOW)



LENGTH 136 bytes

Using the routine

POKE the *y* co-ordinate of the top of the window into location 23296.
POKE the *y* co-ordinate of the bottom of the window into location 23297.
POKE the leftmost PRINT position into location 23298.
POKE the width (number of PRINT positions) into location 23299.

Note: Will not scroll if inappropriate values are used.

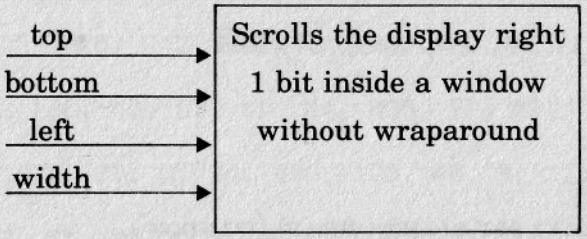
Display: bit left (wrap) (window) data

```
58, 2, 91, 254, 176, 48, 128, 87, 58, 3,

91, 138, 254, 34, 48, 119, 58, 1, 91, 254,

176, 48, 112, 87, 58, 0, 91, 254, 176, 48,

104, 186, 56, 101, 40, 99, 213, 245, 24, 45,

22, 0, 58, 3, 91, 71, 58, 2, 91, 95,

25, 5, 94, 203, 19, 245, 55, 63, 35, 94,

203, 19, 43, 94, 203, 19, 115, 35, 55, 63,

16, 242, 241, 94, 203, 19, 115, 241, 209, 186,
```



```
40, 53, 61, 24, 207, 71, 62, 16, 128, 71,

33, 0, 64, 254, 128, 48, 9, 17, 0, 8,

25, 254, 64, 48, 1, 25, 203, 184, 203, 176,

62, 63, 144, 17, 32, 0, 254, 8, 56, 5,

214, 8, 25, 24, 247, 254, 0, 40, 4, 61,

36, 24, 248, 24, 161, 201
```

DISPLAY: BIT RIGHT (NO WRAP) (WINDOW)



LENGTH 131 bytes

Using the routine

POKE the *y* co-ordinate of the top of the window into location 23296.
POKE the *y* co-ordinate of the bottom of the window into location 23297.
POKE the leftmost PRINT position into location 23298.
POKE the width (number of PRINT positions) into location 23299.

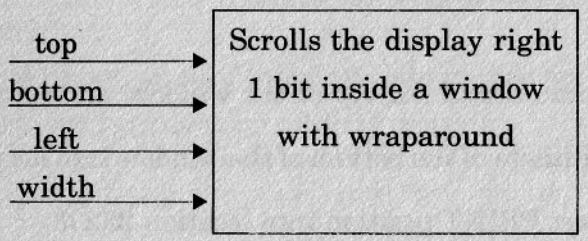
Note: Will not scroll if inappropriate values are used.

Display: bit right (no wrap) (window) data

```
58, 2, 91, 254, 176, 48, 123, 87, 58, 3,

91, 138, 254, 34, 48, 114, 58, 1, 91, 254,

176, 48, 107, 87, 58, 0, 91, 254, 176, 48,

99, 186, 56, 96, 40, 94, 213, 245, 24, 40,
```

```
22, 0, 58, 3, 91, 71, 58, 2, 91, 95,
5, 25, 88, 25, 43, 94, 203, 27, 35, 94,
203, 27, 115, 43, 55, 63, 16, 242, 94, 203,
27, 115, 241, 209, 186, 40, 53, 61, 24, 214,
71, 62, 16, 128, 71, 33, 0, 64, 254, 128,
48, 9, 17, 0, 8, 25, 254, 64, 48, 1,
25, 203, 184, 203, 176, 62, 63, 144, 17, 32,
0, 254, 8, 56, 5, 214, 8, 25, 24, 247,
254, 0, 40, 4, 61, 36, 24, 248, 24, 166,
201
```

DISPLAY: BIT RIGHT (WRAP) (WINDOW)



LENGTH 138 bytes

Using the routine

POKE the *y* co-ordinate of the top of the window into location 23296.
POKE the *y* co-ordinate of the bottom of the window into location 23297.
POKE the leftmost PRINT position into location 23298.
POKE the width (number of PRINT positions) into location 23299.

Note: Will not scroll if inappropriate values are used.

Display: bit right (wrap) (window) data

```
58, 2, 91, 254, 176, 48, 130, 87, 58, 3,
91, 138, 254, 34, 48, 121, 58, 1, 91, 254,
176, 48, 114, 87, 58, 0, 91, 254, 176, 48,
106, 186, 56, 103, 40, 101, 213, 245, 24, 47,
22, 0, 58, 3, 91, 71, 58, 2, 91, 95,
5, 25, 88, 25, 94, 203, 27, 245, 55, 63,
43, 94, 203, 27, 35, 94, 203, 27, 115, 43,
55, 63, 16, 242, 241, 94, 203, 27, 115, 241,
209, 186, 40, 53, 61, 24, 205, 71, 62, 16,
128, 71, 33, 0, 64, 254, 128, 48, 9, 17,
0, 8, 25, 254, 64, 48, 1, 25, 203, 184,
203, 176, 62, 63, 144, 17, 32, 0, 254, 8,
56, 5, 214, 8, 25, 24, 247, 254, 0, 40,
4, 61, 36, 24, 248, 24, 159, 201
```

Example 7.4

Use the cursor keys to scroll the star pattern – but don't forget to press CAPS SHIFT!
Press 's' to stop the program.

```
 10 REM STARGAZING
 20 REM set colours
 30 PAPER 7: INK 0: BORDER 7
 40 CLS
100 REM set new ramtop
110 CLEAR 64774: REM 16K=32006
120 LET start1=64775: REM 16K=3
2007
130 LET start2=start1+138
140 LET start3=start2+136
```

Program 7.4 *(continues)*

```
 150 LET start4=start3+161
 160 LET s=start1
 170 LET length=138+136+161+158
 180 FOR i=1 TO length
 190 READ n
 200 POKE s,n
 210 LET s=s+1
 220 NEXT i
 250 REM dimension arrays
 260 DIM x(10): DIM y(10)
 300 REM set attributes
 310 PAPER 0: INK 7
 320 FOR i=6 TO 15
 330 FOR j=6 TO 25
 340 PRINT AT i,j;" "
 350 NEXT j
 360 NEXT i
 400 REM generate star pattern
 410 FOR i=1 TO 10
 420 READ x(i),y(i)
 430 PLOT x(i),y(i): PLOT x(i)+1
,y(i)+1: PLOT x(i)+1,y(i): PLOT
x(i),y(i)+1
 440 NEXT i
 500 REM scroll
 510 POKE 23296,127: REM top
 520 POKE 23297,48: REM bottom
 530 POKE 23298,6: REM leftmost
 540 POKE 23299,20: REM width
 550 LET k$=INKEY$
 560 IF k$=CHR$ 9 THEN  RANDOMIZ
E USR start1
 570 IF k$=CHR$ 8 THEN  RANDOMIZ
E USR start2
 580 IF k$=CHR$ 11 THEN  RANDOMI
ZE USR start3
 590 IF k$=CHR$ 10 THEN  RANDOMI
ZE USR start4
 600 IF k$<>"s" THEN  GO TO 550
 700 REM finished
 710 PAPER 7: INK 0: BORDER 7
1000 REM DISPLAY  BIT RIGHT (WRA
P) (WINDOW)
1010 DATA 58,2,91,254,176,48,130
,87,58,3
1020 DATA 91,138,254,34,48,121,5
8,1,91,254
1030 DATA 176,48,114,87,58,0,91,
254,176,48
1040 DATA 106,186,56,103,40,101,
213,245,24,47
1050 DATA 22,0,58,3,91,71,58,2,9
1,95
1060 DATA 5,25,88,25,94,203,27,2
45,55,63
```

Program 7.4 *(continues)*


```
1070 DATA 43,94,203,27,35,94,203
,27,115,43
1080 DATA 55,63,16,242,241,94,20
3,27,115,241
1090 DATA 209,186,40,53,61,24,20
5,71,62,16
1100 DATA 128,71,33,0,64,254,128
,48,9,17
1110 DATA 0,8,25,254,64,48,1,25,
203,184
1120 DATA 203,176,62,63,144,17,3
2,0,254,8
1130 DATA 56,5,214,8,25,24,247,2
54,0,40
1140 DATA 4,61,36,24,248,24,159,
201
2000 REM DISPLAY  BIT LEFT (WRAP
) (WINDOW)
2010 DATA 58,2,91,254,176,48,128
,87,58,3
2020 DATA 91,138,254,34,48,119,5
8,1,91,254
2030 DATA 176,48,112,87,58,0,91,
254,176,48
2040 DATA 104,186,56,101,40,99,2
13,245,24,45
2050 DATA 22,0,58,3,91,71,58,2,9
1,95
2060 DATA 25,5,94,203,19,245,55,
63,35,94
2070 DATA 203,19,43,94,203,19,11
5,35,55,63
2080 DATA 16,242,241,94,203,19,1
15,241,209,186
2090 DATA 40,53,61,24,207,71,62,
16,128,71
2100 DATA 33,0,64,254,128,48,9,1
7,0,8
2110 DATA 25,254,64,48,1,25,203,
184,203,176
2120 DATA 62,63,144,17,32,0,254,
8,56,5
2130 DATA 214,8,25,24,247,254,0,
40,4,61
2140 DATA 36,24,248,24,161,201
3000 REM DISPLAY  BIT UP (WRAP)
(WINDOW)
3010 DATA 58,2,91,254,31,48,153,
87,58,3
3020 DATA 91,138,254,34,48,144,5
8,1,91,254
3030 DATA 176,48,137,71,58,0,91,
254,176,48
3040 DATA 129,184,56,126,40,124,
197,14,255,245
```

```
3050 DATA 24,56,229,6,0,58,3,91,
79,17
3060 DATA 4,91,237,176,225,241,1
4,0,61,245
3070 DATA 229,24,35,209,213,229,
6,0,58,3
3080 DATA 91,79,237,176,209,225,
241,193,184,197
3090 DATA 235,32,229,193,58,3,91
,79,6,0
3100 DATA 235,33,4,91,237,176,24
,62,71,62
3110 DATA 16,128,71,33,0,64,254,
128,48,9
3120 DATA 17,0,8,25,254,64,48,1,
25,203
3130 DATA 184,203,176,62,63,144,
17,32,0,254
3140 DATA 8,56,5,214,8,25,24,247
,254,0
3150 DATA 40,4,61,36,24,248,58,2
,91,22
3160 DATA 0,95,25,121,254,255,40
,140,32,159
3170 DATA 201
4000 REM DISPLAY  BIT DOWN (WRAP
) (WINDOW)
4010 DATA 58,2,91,254,31,48,150,
87,58,3
4020 DATA 91,138,254,34,48,141,5
8,0,91,254
4030 DATA 176,48,134,71,58,1,91,
254,176,48
4040 DATA 126,184,48,123,197,14,
255,245,24,55
4050 DATA 229,6,0,58,3,91,79,17,
4,91
4060 DATA 237,176,225,241,14,0,6
0,245,229,24
4070 DATA 34,209,213,229,6,0,58,
3,91,79
4080 DATA 237,176,225,209,241,19
3,184,197,32,230
4090 DATA 193,58,3,91,79,6,0,235
,33,4
4100 DATA 91,237,176,24,62,71,62
,16,128,71
4110 DATA 33,0,64,254,128,48,9,1
7,0,8
4120 DATA 25,254,64,48,1,25,203,
184,203,176
4130 DATA 62,63,144,17,32,0,254,
8,56,5
4140 DATA 214,8,25,24,247,254,0,
40,4,61
```



```
4150 DATA 36,24,248,58,2,91,22,0
,95,25
4160 DATA 121,254,255,40,141,32,
160,201
5000 REM STAR POSITIONS
5010 DATA 63,60,80,80,56,96,58,1
01,103,112,143,104,160,72,185,97
,192,64,152,50
```

Program 7.4

Example 7.5

The following program (7.5) uses three different kinds of scrolling routines to provide an effective display showing a fleet of ships sailing across the screen, then behind an island.

```
  10 REM SHIP
  20 REM set colours
  30 PAPER 7: INK 0: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65068: REM 16K=32300
 120 LET start1=65069: REM 16K=3
2301: REM BIT DOWN (WRAP)
 130 LET start2=start1+129: REM
DISPLAY  BIT LEFT (WRAP) (WINDOW
)
 140 LET start3=start2+136: REM
FAST BIT RIGHT (WRAP)
 150 LET s=start1
 160 LET length=129+136+34
 170 FOR i=1 TO length
 180 READ n
 190 POKE s,n
 200 LET s=s+1
 210 NEXT i
 250 REM define window variables
 260 LET wt=23296
 270 LET wb=23297
 280 LET wl=23298
 290 LET ww=23299
 300 REM define UDG characters
 310 FOR i=1 TO 10
 320 READ c$
 330 FOR j=0 TO 7
 340 READ d
 350 POKE USR c$+j,d
 360 NEXT j
 370 NEXT i
 400 REM set attributes for line
s 0-7
 410 FOR l=0 TO 7
 420 FOR c=0 TO 31
```

Program 7.5 *(continues)*

```
 430 PRINT  PAPER 5; INK 7;AT l,
c;CHR$ 32
 440 NEXT c
 450 NEXT l
 500 REM set attributes for line
s 8-15
 510 FOR l=8 TO 15
 520 FOR c=0 TO 31
 530 PRINT  PAPER 5; INK 0;AT l,
c;CHR$ 32
 540 NEXT c
 550 NEXT l
 600 REM set attributes for line
s 16-21
 610 FOR l=16 TO 21
 620 FOR c=0 TO 31
 630 PRINT  PAPER 1; INK 7;AT l,
c;CHR$ 32
 640 NEXT c
 650 NEXT l
 700 REM print clouds
 710 PAPER 5: INK 7
 720 PRINT AT 2,26;CHR$ 146+CHR$
 147+CHR$ 148
 730 PRINT AT 1,27;CHR$ 145
 740 PRINT AT 6,17;CHR$ 146+CHR$
 147+CHR$ 148
 750 PRINT AT 5,18;CHR$ 145
 760 PRINT AT 3,7;CHR$ 146+CHR$
147+CHR$ 148
 770 PRINT AT 2,8;CHR$ 145
 800 REM print waves
 810 PAPER 1: INK 7
 820 RANDOMIZE
 830 FOR l=16 TO 21
 840 FOR c=0 TO 31
 850 IF RND>=0.5 THEN  PRINT AT
l,c;CHR$ 144
 860 NEXT c
 870 NEXT l
 900 REM print ship
 910 PAPER 5: INK 0
 920 PRINT AT 15,29;CHR$ 149+CHR
$ 150+CHR$ 151
1000 REM print land and trees
1010 PRINT  INK 7;AT 15,0;CHR$ 1
43+CHR$ 143+CHR$ 143+CHR$ 143+CH
R$ 143
1020 PRINT  INK 7;AT 14,0;CHR$ 1
43+CHR$ 143+CHR$ 143+CHR$ 143+CH
R$ 153
1030 PRINT  INK 4;AT 13,0;CHR$ 1
43+CHR$ 153
1040 PRINT  INK 4;AT 12,0;CHR$ 1
52
```

Program 7.5 *(continues)*


```
1050 PRINT  INK 4;AT 13,2;CHR$ 1
52+CHR$ 152
1100 REM execute scrolls
1110 POKE wl,5: POKE ww,27
1120 FOR i=1 TO 1000
1130 POKE wt,47: POKE wb,0
1140 RANDOMIZE USR start1
1150 POKE wt,63: POKE wb,48
1160 RANDOMIZE USR start2
1170 RANDOMIZE USR start3
1180 NEXT i
2000 REM DISPLAY  BIT DOWN (WRAP
)
2010 DATA 58,0,91,254,176,48,121
,71,58,1
2020 DATA 91,254,176,48,113,184,
48,110,40,108
2030 DATA 197,14,255,245,24,47,2
29,1,32,0
2040 DATA 17,4,91,237,176,225,24
1,14,0,60
2050 DATA 245,229,24,29,209,213,
229,1,32,0
2060 DATA 237,176,209,225,241,19
3,184,197,235,32
2070 DATA 232,193,1,32,0,235,33,
4,91,237
2080 DATA 176,24,55,71,62,16,128
,71,33,0
2090 DATA 64,254,128,48,9,17,0,8
,25,254
2100 DATA 64,48,1,25,203,184,203
,176,62,63
2110 DATA 144,17,32,0,254,8,56,5
,214,8
2120 DATA 25,24,247,254,0,40,4,6
1,36,24
2130 DATA 246,121,254,255,40,156
,32,172,201
3000 REM DISPLAY  BIT LEFT (WRAP
) (WINDOW)
3010 DATA 58,2,91,254,176,48,128
,87,58,3
3020 DATA 91,138,254,34,48,119,5
8,1,91,254
3030 DATA 176,48,112,87,58,0,91,
254,176,48
3040 DATA 104,186,56,101,40,99,2
13,245,24,45
3050 DATA 22,0,58,3,91,71,58,2,9
1,95
3060 DATA 25,5,94,203,19,245,55,
63,35,94
3070 DATA 203,19,43,94,203,19,11
5,35,55,63
```

```
3080 DATA 16,242,241,94,203,19,1
15,241,209,186
3090 DATA 40,53,61,24,207,71,62,
16,128,71
3100 DATA 33,0,64,254,128,48,9,1
7,0,8
3110 DATA 25,254,64,48,1,25,203,
184,203,176
3120 DATA 62,63,144,17,32,0,254,
8,56,5
3130 DATA 214,8,25,24,247,254,0,
40,4,61
3140 DATA 36,24,248,24,161,201
4000 REM DISPLAY  FAST BIT RIGHT
 (WRAP)
4010 DATA 33,255,71,6,64,197,6,3
1,94,203
4020 DATA 27,245,43,94,203,27,35
,94,203,27
4030 DATA 115,43,16,244,241,94,2
03,27,115,43
4040 DATA 193,16,228,201
5000 REM UDG DEFINITIONS
5010 REM wave
5020 DATA "A",0,0,0,16,16,40,199
,0
5030 REM cloud
5040 DATA "B",0,0,0,0,0,60,126,2
55
5050 DATA "C",1,15,63,127,63,31,
15,0
5060 DATA "D",255,255,255,255,25
5,255,62,0
5070 DATA "E",192,248,254,254,25
4,252,120,0
5080 REM ship
5090 DATA "F",0,0,0,0,255,109,63
,31
5100 DATA "G",6,6,255,255,255,18
2,255,255
5110 DATA "H",192,192,224,224,25
5,219,255,255
5120 REM tree
5130 DATA "I",62,90,172,42,8,8,8
,255
5140 REM cliff edge
5150 DATA "J",128,192,240,252,25
4,254,255,255
```

Program 7.5

After running the program, try improving the effect produced by moving the waves. This can be done by moving the waves down (as at present) and then to the left (or right). The existing DISPLAY:

BIT LEFT (WRAP) (WINDOW) can be used, or the DISPLAY: FAST BIT RIGHT (WRAP) routine could be employed.

Attribute scrolling

The attributes may be scrolled in a similar way to the display file. However, because of the different construction of the attribute file, scrolling in any direction is only possible in 1 byte units. This makes attribute scrolling suitable for use with any of the byte scrolling display routines. The appropriate attribute scrolling routine may be executed after a display file byte scroll, and will provide movement of the attributes to accompany the movement of the display file, resulting in a scrolling movement which moves not only the pixels, but their accompanying colours. The attributes may be scrolled independently of the display file, of course, and this can be used to provide some rather interesting effects.

All the attribute scrolling routines are wraparound movements of the attributes, since scrolling without wraparound does not normally have any application, where the attributes are concerned. As with the display file scrolling routines, attribute scrolling may take place over the full screen, or inside a window. When setting the limits for the window, PRINT positions are used in the vertical as well as the horizontal direction, because these correspond with the resolution available inside the attribute file.

FULL SCREEN SCROLLING

ATTRIBUTES: LEFT

Scrolls the attribute file
1 byte left over
the full screen

LENGTH 23 bytes

Using the routine

No values need be set before executing this routine.

Note: The 20th data item controls the number of lines scrolled, counting from the top of the screen, so this may be set to 24 to include the bottom two lines.

Attributes: left data

33, 0, 88, 62, 0, 6, 31, 86, 35, 94,
43, 115, 35, 16, 249, 114, 35, 60, 254, 22*,
32, 239, 201

ATTRIBUTES: RIGHT

Scrolls the attribute file 1 byte right over the full screen

LENGTH 23 bytes

Using the routine
No values need be set before executing this routine.

Note: To include the bottom two lines, change the 2nd data item to 255, and the 20th data item to 24.

Attributes: right data

33, 191*, 90, 62, 0, 6, 31, 86, 43, 94,
35, 115, 43, 16, 249, 114, 43, 60, 254, 22*,
32, 239, 201

ATTRIBUTES: UP

Scrolls the attribute file upwards 1 byte over the full screen

LENGTH 36 bytes

Using the routine
No values need be set before executing this routine.

Note: The 7th data item controls the number of lines scrolled, so this may be changed to 23, to include the bottom two rows of the screen. The 33rd data item controls the number of columns scrolled, and this may be changed to a number between 0 and 32.

Attributes: up data

33, 0, 88, 62, 0, 6, 21*, 229, 86, 229,
213, 17, 32, 0, 25, 209, 94, 225, 115, 213,
17, 32, 0, 25, 209, 16, 238, 114, 225, 35,
60, 254, 32*, 32, 226, 201

ATTRIBUTES: DOWN

Scrolls the attribute file 1 byte downwards over the full screen

LENGTH 42 bytes

Using the routine
No values need be set before using this routine.

Attributes: down data

33, 160, 90, 62, 0, 6, 21, 229, 86, 229,
55, 63, 213, 17, 32, 0, 237, 82, 209, 94,
225, 115, 55, 63, 213, 17, 32, 0, 237, 82,
209, 16, 232, 114, 225, 35, 60, 254, 32, 32,
220, 201

Example 7.6

As in Example 7.4, the cursor keys are used to control the scrolling. Pressing 's' stops the program.

```
 10 REM COLOURED MESSAGE
 20 REM set colours
 30 PAPER 7: INK 0: BORDER 7
 40 CLS
100 REM set new ramtop
110 CLEAR 65243: REM 16K=32475
120 LET start1=65244: REM 16K=3
2476
130 LET start2=start1+23
140 LET start3=start2+23
150 LET start4=start3+36
160 LET s=start1
170 LET length=23+23+36+42
180 FOR i=1 TO length
190 READ n
200 POKE s,n
```

Program 7.6 *(continues)*

```
 210 LET s=s+1
 220 NEXT i
 300 REM set up attributes
 310 LET ink=0
 320 FOR i=0 TO 21
 330 FOR j=0 TO 31
 340 PRINT  INK ink;AT i,j;" "
 350 LET ink=ink+1
 360 IF ink=7 THEN  LET ink=0
 370 NEXT j
 380 NEXT i
 400 REM print message
 410 PRINT AT 3,8;"PRESS CAPS SH
IFT"
 420 PRINT AT 8,7;"AND THE CURSO
R KEYS"
 430 PRINT AT 13,9;"AND WATCH TH
E"
 440 PRINT AT 18,6;"PRETTY COLOU
RS CHANGE."
 500 REM scroll
 510 LET k$=INKEY$
 520 IF k$=CHR$ 9 THEN  RANDOMIZ
E USR start1
 530 IF k$=CHR$ 8 THEN  RANDOMIZ
E USR start2
 540 IF k$=CHR$ 11 THEN  RANDOMI
ZE USR start3
 550 IF k$=CHR$ 10 THEN  RANDOMI
ZE USR start4
 560 IF k$<>"s" THEN  GO TO 510
 600 REM finished
1000 REM ATTRIBUTES RIGHT
1010 DATA 33,191,90,62,0,6,31,86
,43,94
1020 DATA 35,115,43,16,249,114,4
3,60,254,22
1030 DATA 32,239,201
2000 REM ATTRIBUTES LEFT
2010 DATA 33,0,88,62,0,6,31,86,3
5,94
2020 DATA 43,115,35,16,249,114,3
5,60,254,22
2030 DATA 32,239,201
3000 REM ATTRIBUTES UP
3010 DATA 33,0,88,62,0,6,21,229,
86,229
3020 DATA 213,17,32,0,25,209,94,
225,115,213
3030 DATA 17,32,0,25,209,16,238,
114,225,35
3040 DATA 60,254,32,32,226,201
4000 REM ATTRIBUTES DOWN
4010 DATA 33,160,90,62,0,6,21,22
9,86,229
```



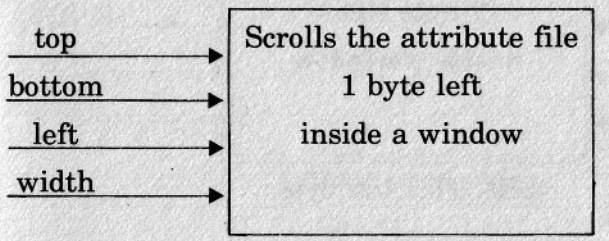
```
4020 DATA 55,63,213,17,32,0,237,
82,209,94
4030 DATA 225,115,55,63,213,17,3
2,0,237,82
4040 DATA 209,16,232,114,225,35,
60,254,32,32
4050 DATA 220,201
```

Program 7.6

THE SCROLLING WINDOW AND THE ATTRIBUTES

The attribute scrolling routines can be modified to recognize the existence of a window on the screen, in just the same way that the display scrolling routines were modified to provide new routines. These new attribute scrolling routines incorporate the same type of safety checks as the display scrolling routines, returning to BASIC safely, without scrolling if any of the values describing the window do not make sense. This allows the attribute scrolling routines to be used with some degree of safety.

ATTRIBUTES: LEFT (WINDOW)



LENGTH 97 bytes

Using the routine

POKE the top PRINT row into location 23296.
POKE the bottom PRINT row into location 23297.
POKE the leftmost PRINT position into location 23298.
POKE the width (number of PRINT positions) into location 23299.

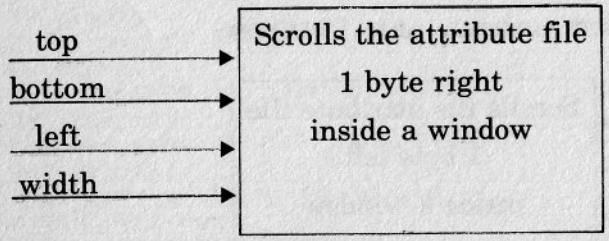
Note: Will not scroll if inappropriate values are used.

Attributes: left (window) data

```
58, 0, 91, 254, 22, 48, 89, 87, 58, 1,

91, 254, 22, 48, 81, 186, 40, 78, 56, 76,
```

```
58, 2, 91, 254, 32, 48, 69, 87, 58, 3,

91, 138, 254, 34, 48, 60, 33, 0, 88, 58,

2, 91, 95, 22, 0, 25, 17, 32, 0, 58,

0, 91, 254, 0, 40, 4, 25, 61, 24, 248,

58, 0, 91, 95, 58, 1, 91, 60, 147, 71,

197, 229, 86, 58, 3, 91, 61, 35, 94, 43,

115, 35, 61, 254, 0, 32, 246, 114, 225, 17,

32, 0, 25, 193, 16, 230, 201
```

ATTRIBUTES: RIGHT (WINDOW)



LENGTH 105 bytes

Using the routine

POKE the top PRINT row into location 23296.
POKE the bottom PRINT row into location 23297.
POKE the leftmost PRINT position into location 23298.
POKE the width (number of PRINT positions) into location 23299.

Note: Will not scroll if inappropriate values are used.

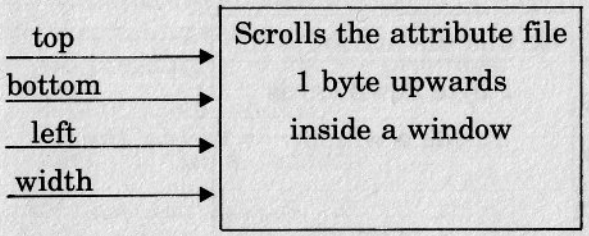
Attributes: right (window) data

```
58, 0, 91, 254, 22, 48, 93, 87, 58, 1,

91, 254, 22, 48, 85, 186, 40, 82, 56, 80,

58, 2, 91, 254, 32, 48, 73, 87, 58, 3,
```



```
91, 138, 254, 34, 48, 64, 33, 0, 88, 58,

2, 91, 95, 22, 0, 25, 58, 3, 91, 61,

22, 0, 95, 25, 17, 32, 0, 58, 0, 91,

254, 0, 40, 4, 25, 61, 24, 248, 58, 0,

91, 95, 58, 1, 91, 60, 147, 71, 197, 229,

86, 58, 3, 91, 61, 43, 94, 35, 115, 43,

61, 254, 0, 32, 246, 114, 225, 17, 32, 0,

25, 193, 16, 230, 201
```

ATTRIBUTES: UP (WINDOW)



LENGTH 102 bytes

Using the routine

POKE the top PRINT row into location 23296.
POKE the bottom PRINT row into location 23297.
POKE the leftmost PRINT position into location 23298.
POKE the width (number of PRINT positions) into location 23299.

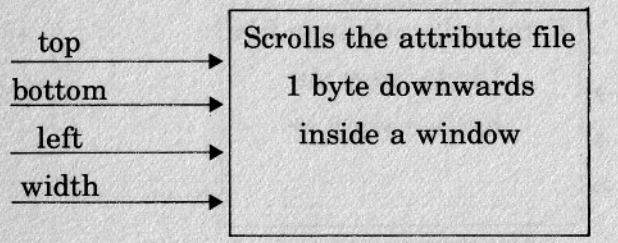
Note: Will not scroll if inappropriate values are used.

Attributes: up (window) data

```
58, 0, 91, 254, 22, 48, 94, 87, 58, 1,

91, 254, 22, 48, 86, 186, 40, 83, 56, 81,

58, 2, 91, 254, 32, 48, 74, 87, 58, 3,
```

```
91, 138, 254, 34, 48, 65, 33, 0, 88, 58,

2, 91, 95, 22, 0, 25, 17, 32, 0, 58,

0, 91, 254, 0, 40, 4, 25, 61, 24, 248,

58, 0, 91, 95, 58, 1, 91, 147, 79, 58,

3, 91, 65, 229, 86, 229, 213, 17, 32, 0,

25, 209, 94, 225, 115, 213, 17, 32, 0, 25,

209, 16, 238, 114, 225, 35, 61, 254, 0, 32,

227, 201
```

ATTRIBUTES: DOWN (WINDOW)



LENGTH 108 bytes

Using the routine

POKE the top PRINT row into location 23296.
POKE the bottom PRINT row into location 23297.
POKE the leftmost PRINT position into location 23298.
POKE the width (number of PRINT positions) into location 23299.

Note: Will not scroll if inappropriate values are used.

Attributes: down (window) data

```
58, 0, 91, 254, 22, 48, 100, 87, 58, 1,

91, 254, 22, 48, 92, 186, 40, 89, 56, 87,

58, 2, 91, 254, 32, 48, 80, 87, 58, 3,
```



```
91, 138, 254, 34, 48, 71, 33, 0, 88, 58,

2, 91, 95, 22, 0, 25, 17, 32, 0, 58,

1, 91, 254, 0, 40, 4, 25, 61, 24, 248,

58, 0, 91, 95, 58, 1, 91, 147, 79, 58,

3, 91, 65, 229, 86, 229, 55, 63, 213, 17,

32, 0, 237, 82, 209, 94, 225, 115, 55, 63,

213, 17, 32, 0, 237, 82, 209, 16, 232, 114,

225, 35, 61, 254, 0, 32, 221, 201
```

Example 7.7

This example illustrates the use of a scrolling window with the attribute scrolling routines. Scrolling is controlled using the cursor keys, and pressing 's' stops the program.

```
  10 REM COLOURED WINDOW
  20 REM set colours
  30 PAPER 7: INK 0: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 64955: REM 16K=32187
 120 LET start1=64956: REM 16K=3
2188
 130 LET start2=start1+105
 140 LET start3=start2+97
 150 LET start4=start3+102
 160 LET s=start1
 170 LET length=105+97+102+108
 180 FOR i=1 TO length
 190 READ n
 200 POKE s,n
 210 LET s=s+1
 220 NEXT i
 250 REM define message
 260 LET m$="THE SCROLLING WINDO
W"
 300 REM set up attributes
 310 LET ink=0: LET paper=7
 320 FOR i=8 TO 12
 330 LET paper=paper-1
 340 FOR j=6 TO 25
 350 PRINT  PAPER paper; INK ink
;AT i,j;m$(j-5)
```

Program 7.7 *(continues)*

```
360 LET ink=ink+1
370 IF ink=7 THEN  LET ink=0
380 NEXT j
390 NEXT i
400 REM define window
410 POKE 23296,8: REM top
420 POKE 23297,12
430 POKE 23298,6: REM leftmost
440 POKE 23299,20: REM width
500 REM scroll
510 LET k$=INKEY$
520 IF k$=CHR$ 9 THEN  RANDOMIZ
E USR start1
530 IF k$=CHR$ 8 THEN  RANDOMIZ
E USR start2
540 IF k$=CHR$ 11 THEN  RANDOMI
ZE USR start3
550 IF k$=CHR$ 10 THEN  RANDOMI
ZE USR start4
560 IF k$<>"s" THEN  GO TO 510
600 REM finished
1000 REM ATTRIBUTES RIGHT (WINDO
W)
1010 DATA 58,0,91,254,22,48,93,8
7,58,1
1020 DATA 91,254,22,48,85,186,40
,82,56,80
1030 DATA 58,2,91,254,32,48,73,8
7,58,3
1040 DATA 91,138,254,34,48,64,33
,0,88,58
1050 DATA 2,91,95,22,0,25,58,3,9
1,61
1060 DATA 22,0,95,25,17,32,0,58,
0,91
1070 DATA 254,0,40,4,25,61,24,24
8,58,0
1080 DATA 91,95,58,1,91,60,147,7
1,197,229
1090 DATA 86,58,3,91,61,43,94,35
,115,43
1100 DATA 61,254,0,32,246,114,22
5,17,32,0
1110 DATA 25,193,16,230,201
2000 REM ATTRIBUTES LEFT (WINDOW
)
2010 DATA 58,0,91,254,22,48,89,8
7,58,1
2020 DATA 91,254,22,48,81,186,40
,78,56,76
2030 DATA 58,2,91,254,32,48,69,8
7,58,3
2040 DATA 91,138,254,34,48,60,33
,0,88,58
2050 DATA 2,91,95,22,0,25,17.32,
0,58
2060 DATA 0,91,254,0,40,4,25,61,
```

Program 7.7 *(continues)*

```
24,248
2070 DATA 58,0,91,95,58,1,91,60,
147,71
2080 DATA 197,229,86,58,3,91,61,
35,94,43
2090 DATA 115,35,61,254,0,32,246
,114,225,17
2100 DATA 32,0,25,193,16,230,201
3000 REM ATTRIBUTES UP (WINDOW)
3010 DATA 58,0,91,254,22,48,94,8
7,58,1
3020 DATA 91,254,22,48,86,186,40
,83,56,81
3030 DATA 58,2,91,254,32,48,74,8
7,58,3
3040 DATA 91,138,254,34,48,65,33
,0,88,58
3050 DATA 2,91,95,22,0,25,17,32,
0,58
3060 DATA 0,91,254,0,40,4,25,61,
24,248
3070 DATA 58,0,91,95,58,1,91,147
,79,58
3080 DATA 3,91,65,229,86,229,213
,17,32,0
3090 DATA 25,209,94,225,115,213,
17,32,0,25
3100 DATA 209,16,238,114,225,35,
61,254,0,32
3110 DATA 227,201
4000 REM ATTRIBUTES DOWN (WINDOW
)
4010 DATA 58,0,91,254,22,48,100,
87,58,1
4020 DATA 91,254,22,48,92,186,40
,89,56,87
4030 DATA 58,2,91,254,32,48,80,8
7,58,3
4040 DATA 91,138,254,34,48,71,33
,0,88,58
4050 DATA 2,91,95,22,0,25,17,32,
0,58
4060 DATA 1,91,254,0,40,4,25,61,
24,248
4070 DATA 58,0,91,95,58,1,91,147
,79,58
4080 DATA 3,91,65,229,86,229,55,
63,213,17
4090 DATA 32,0,237,82,209,94,225
,115,55,63
4100 DATA 213,17,32,0,237,82,209
,16,232,114
4110 DATA 225,35,61,254,0,32,221
,201
```

Program 7.7

Titlescroll

TITLESCROLL uses the display created by the TITLETYPE program at the end of Chapter 1, and creates a moving message on the screen. Four scrolling routines are used to allow bit-by-bit movement up, down, left, and right. The title created by TITLETYPE occupies a known position on the screen, so the display can be handled easily. Changing the position or size of the message created by TITLETYPE may mean changing some of the values in TITLESCROLL.

When scrolling begins, two windows are created on the screen in turn, and scrolling takes place upwards in the top window, and downwards in the bottom window. When the contents of the windows have been scrolled 336 times, the size of the windows is changed, and horizontal scrolling begins.

Since two windows cannot exist simultaneously, it is necessary to switch between the two inside the loops which control the scrolling. However, this does not spoil the illusion, since switching is accomplished fairly quickly.

When TITLESCROLL is RUN, it will attempt to load the display created previously by the TITLETYPE program, so this should be available on tape.

The basic technique used in this program (7.A) can easily be modified and used with other screen displays.

```
 10 REM TITLESCROLL
 20 REM set colours
 30 PAPER 7: INK 0: BORDER 7
 40 CLS
100 REM set new ramtop
110 CLEAR 64774: REM 16K=32006
120 LET start1=64775: REM 16K=3
2007
130 LET start2=start1+138
140 LET start3=start2+136
150 LET start4=start3+161
160 LET s=start1
170 LET length=138+136+161+158
180 FOR i=1 TO length
190 READ n
200 POKE s,n
210 LET s=s+1
220 NEXT i
```

Program 7.A *(continues)*


```
250 REM define window
260 LET t=23296
270 LET b=23297
280 LET l=23298
290 LET w=23299
300 REM set attributes
310 PAPER 5: INK 2: BORDER 5: C
LS
400 REM load display file
410 LOAD ""CODE 16384
500 REM up and down
510 FOR i=1 TO 56*6
520 POKE t,144: POKE b,88: POKE
 l,8: POKE w,16: RANDOMIZE USR s
tart3
530 POKE t,87: POKE b,32: POKE
l,2: POKE w,28: RANDOMIZE USR st
art4
540 NEXT i
545 POKE l,2: POKE w,28
550 FOR i=1 TO 28*8*3
560 POKE t,136: POKE b,94: RAND
OMIZE USR start1
570 POKE t,72: POKE b,48: RANDO
MIZE USR start2
580 NEXT i
700 REM finished
710 PAPER 7: INK 0
1000 REM DISPLAY  BIT RIGHT (WRA
P) (WINDOW)
1010 DATA 58,2,91,254,176,48,130
,87,58,3
1020 DATA 91,138,254,34,48,121,5
8,1,91,254
1030 DATA 176,48,114,87,58,0,91,
254,176,48
1040 DATA 106,186,56,103,40,101,
213,245,24,47
1050 DATA 22,0,58,3,91,71,58,2,9
1,95
1060 DATA 5,25,88,25,94,203,27,2
45,55,63
1070 DATA 43,94,203,27,35,94,203
,27,115,43
1080 DATA 55,63,16,242,241,94,20
3,27,115,241
1090 DATA 209,186,40,53,61,24,20
5,71,62,16
1100 DATA 128,71,33,0,64,254,128
,48,9,17
1110 DATA 0,8,25,254,64,48,1,25,
203,184
1120 DATA 203,176,62,63,144,17,3
2,0,254,8
1130 DATA 56,5,214,8,25,24,247,2
```

```
54,0,40
1140 DATA 4,61,36,24,248,24,159,
201
2000 REM DISPLAY  BIT LEFT (WRAP
) (WINDOW)
2010 DATA 58,2,91,254,176,48,128
,87,58,3
2020 DATA 91,138,254,34,48,119,5
8,1,91,254
2030 DATA 176,48,112,87,58,0,91,
254,176,48
2040 DATA 104,186,56,101,40,99,2
13,245,24,45
2050 DATA 22,0,58,3,91,71,58,2,9
1,95
2060 DATA 25,5,94,203,19,245,55,
63,35,94
2070 DATA 203,19,43,94,203,19,11
5,35,55,63
2080 DATA 16,242,241,94,203,19,1
15,241,209,186
2090 DATA 40,53,61,24,207,71,62,
16,128,71
2100 DATA 33,0,64,254,128,48,9,1
7,0,8
2110 DATA 25,254,64,48,1,25,203,
184,203,176
2120 DATA 62,63,144,17,32,0,254,
8,56,5
2130 DATA 214,8,25,24,247,254,0,
40,4,61
2140 DATA 36,24,248,24,161,201
3000 REM DISPLAY  BIT UP (WRAP)
(WINDOW)
3010 DATA 58,2,91,254,31,48,153,
87,58,3
3020 DATA 91,138,254,34,48,144,5
8,1,91,254
3030 DATA 176,48,137,71,58,0,91,
254,176,48
3040 DATA 129,184,56,126,40,124,
197,14,255,245
3050 DATA 24,56,229,6,0,58,3,91,
79,17
3060 DATA 4,91,237,176,225,241,1
4,0,61,245
3070 DATA 229,24,35,209,213,229,
6,0,58,3
3080 DATA 91,79,237,176,209,225,
241,193,184,197
3090 DATA 235,32,229,193,58,3,91
,79,6,0
3100 DATA 235,33,4,91,237,176,24
,62,71,62
3110 DATA 16,128,71,33,0,64,254,
```



```
128,48,9
3120 DATA 17,0,8,25,254,64,48,1,
25,203
3130 DATA 184,203,176,62,63,144,
17,32,0,254
3140 DATA 8,56,5,214,8,25,24,247
,254,0
3150 DATA 40,4,61,36,24,248,58,2
,91,22
3160 DATA 0,95,25,121,254,255,40
,140,32,159
3170 DATA 201
4000 REM DISPLAY  BIT DOWN (WRAP
) (WINDOW)
4010 DATA 58,2,91,254,31,48,150,
87,58,3
4020 DATA 91,138,254,34,48,141,5
8,0,91,254
4030 DATA 176,48,134,71,58,1,91,
254,176,48
4040 DATA 126,184,48,123,197,14,
255,245,24,55
4050 DATA 229,6,0,58,3,91,79,17,
4,91
4060 DATA 237,176,225,241,14,0,6
0,245,229,24
4070 DATA 34,209,213,229,6,0,58,
3,91,79
4080 DATA 237,176,225,209,241,19
3,184,197,32,230
4090 DATA 193,58,3,91,79,6,0,235
,33,4
4100 DATA 91,237,176,24,62,71,62
,16,128,71
4110 DATA 33,0,64,254,128,48,9,1
7,0,8
4120 DATA 25,254,64,48,1,25,203,
184,203,176
4130 DATA 62,63,144,17,32,0,254,
8,56,5
4140 DATA 214,8,25,24,247,254,0,
40,4,61
4150 DATA 36,24,248,58,2,91,22,0
,95,25
4160 DATA 121,254,255,40,141,32,
160,201
```

Program 7.A

8 AN 'ARCADE GRAPHICS' SYSTEM

When a user-defined graphics character is created, the shape of the character is defined as an 8 × 8 pattern of lit and unlit pixels. UDG characters suffer from two related disadvantages.

1. The maximum size of a single character is 64 pixels.
2. The minimum size of a single character is 64 pixels.

Using several UDG characters to produce a single shape is possible, but does tend to increase the complexity of the routines which print the characters, and slows the printing process.

The ROTATED SCALED CHARACTER PLOTTER routine may be used to enlarge a single UDG character, but this means that each single pixel in the original character definition is expanded to occupy a larger area. Imagine a small square, as shown in Fig. 8.1, defined inside a single UDG character.

Figure 8.1 A square inside a UDG character

Figure 8.2 shows that if this character is plotted using a scale of 2, not only do the sides of the square increase in length, they also increase in thickness.

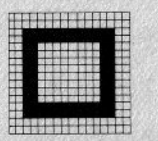


Figure 8.2 The square when plotted using a scale of 2

This effect may not be acceptable when creating line drawings, so a different system must be used. This alternative system is based on the use of a series of shape description codes.

Now, instead of defining the pattern of lit and unlit pixels for a single character space, a series of codes are stored which describe

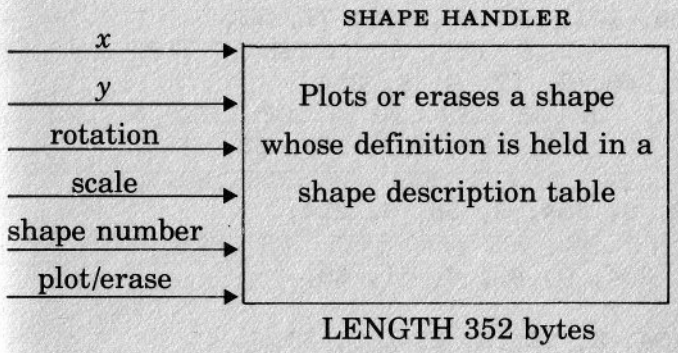
how to construct the shape. The only restriction on the maximum size of the shape is the physical size of the screen. This system allows complex shapes to be manipulated with ease. Together, the use of UDG characters and the shape description system make possible the creation of arcade-quality graphics on the Spectrum. So, for any particular application, an arcade graphics system can be assembled using a selection of UDG and shape description routines.

The shape description system offers a number of facilities which may be used when manipulating one or more objects.
Each shape may be:

(a) drawn
(b) erased
(c) rotated before being drawn
(d) scaled before being drawn.

In addition

(e) several shape descriptions may be held in memory at the same time;
(f) a collision flag is provided.



USING THE ROUTINE

POKE the *x* co-ordinate of the start of the shape into location 23296.
POKE the *y* co-ordinate of the start of the shape into location 23297.
POKE the rotation (0–7 multiples of 45°) into location 23298.
POKE the scale (1–255) into location 23299.
POKE the shape number into location 23300.
POKE 0 (ERASE) or 1 (PLOT) into location 23301.

Note: The routine uses locations 23303 and 23304 as workspace. The shape definitions should be stored in locations 23305 to 23551. A collision flag is provided in location 23302.

Shape handler data

```
62, 0, 50, 6, 91, 33, 0, 91, 126, 35,

70, 17, 6, 0, 25, 119, 120, 35, 119, 33,

8, 91, 14, 1, 58, 4, 91, 71, 254, 1,

40, 11, 35, 126, 254, 69, 32, 250, 12, 121,

184, 32, 245, 229, 245, 241, 225, 35, 126, 254,

80, 40, 8, 254, 78, 32, 93, 55, 63, 24,

1, 55, 35, 229, 245, 58, 3, 91, 95, 241,

213, 245, 48, 9, 33, 7, 91, 78, 35, 70,

62, 175, 184, 56, 121, 62, 16, 128, 71, 33,

0, 64, 254, 128, 48, 9, 17, 0, 8, 25,

254, 64, 48, 1, 25, 203, 184, 203, 176, 62,

63, 144, 17, 32, 0, 254, 8, 56, 5, 214,

8, 25, 24, 247, 254, 0, 40, 4, 61, 36,

24, 248, 121, 254, 8, 56, 5, 214, 8, 35,

24, 247, 79, 62, 7, 145, 24, 8, 24, 151,

24, 110, 24, 110, 24, 174, 55, 63, 87, 71,

94, 254, 0, 40, 4, 203, 11, 16, 252, 203,

67, 40, 5, 62, 1, 50, 6, 91, 241, 245,

48, 13, 58, 5, 91, 254, 1, 32, 4, 203,

195, 24, 2, 203, 131, 122, 66, 254, 0, 40,

4, 203, 3, 16, 252, 115, 241, 209, 225, 229,
```



```
213, 245, 126, 254, 85, 32, 4, 22, 1, 24,

22, 254, 68, 32, 4, 22, 5, 24, 14, 254,

82, 32, 4, 22, 3, 24, 6, 254, 76, 32,

23, 22, 7, 58, 2, 91, 254, 8, 48, 14,

130, 254, 9, 56, 4, 214, 8, 24, 248, 87,

24, 8, 24, 87, 24, 79, 24, 142, 24, 134,

58, 7, 91, 79, 58, 8, 91, 71, 122, 254,

1, 32, 1, 4, 254, 2, 32, 2, 4, 12,

254, 3, 32, 1, 12, 254, 4, 32, 2, 12,

5, 254, 5, 32, 1, 5, 254, 6, 32, 2,

5, 13, 254, 7, 32, 1, 13, 254, 8, 32,

2, 13, 4, 121, 50, 7, 91, 120, 50, 8,

91, 241, 209, 245, 29, 123, 254, 0, 40, 184,

241, 213, 245, 24, 177, 241, 209, 24, 1, 241,

225, 201
```

Shape descriptions

A shape description consists of a series of codes.
There are three types of code which may be used within a description.

1. A TASK code describing what to do with the current pixel.
   P = plot the current pixel.
   N = do not plot the current pixel.
2. A MOVE code describing how to move to the next pixel.
   R = move right.
   L = move left.
   U = move up.
   D = move down.

3. An END OF SHAPE code.
   E = end of shape.

Once a description has been devised, the appropriate codes should be POKEd into a shape description table beginning at location 23305. Here are some simple shape descriptions, together with details of how to manipulate the shapes once they have been described.

SQUARE

The shape description for a square would be devised as shown in Fig. 8.3.

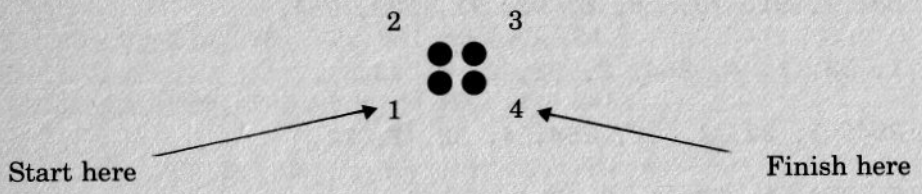


Figure 8.3 Pixels which define a square
(● denotes a pixel which is to be plotted)

Pixel 1 is to be Plotted, then move Up to pixel 2.
Pixel 2 is to be Plotted, then move Right to pixel 3.
Pixel 3 is to be Plotted, then move Down to pixel 4.
Pixel 4 is to be Plotted, then move Left to pixel 1.
End of shape description.

Note that, after plotting pixel 4, a direction is specified. This has two effects.

First, it implies that movement takes place in the 4→1 direction. This is not important if the shape is to be plotted with scale = 1, but it is very important if any other scale is to be used. Scaling takes place by repeating each pair of TASK + MOVE instructions a number of times. To ensure that the square remains a closed four-sided figure when scaled, the direction of motion after plotting pixel 4 is very important.

The second effect of specifying a direction after plotting pixel 4 is to place the E (end of description) code in the position a TASK code would normally occupy. This is very important.

The codes for the square are:

P U P R P D P L E

Shape description tables should be stored in locations 23305 onwards, and this can be done using the following technique:


```
LET s$ = "PUPRPDPLE"
LET s = 23305
FOR c = 1 TO LEN s$
POKE s,CODE s$(c)
LET s = s + 1
NEXT c
```

Important: It is essential that the shape description codes are stored as CAPITAL LETTERS.

The six lines of program above place the numeric codes corresponding to the capital letters forming the shape description for the square into locations 23305–23313.

To draw the square, set the values of *x* and *y* (to set the position of pixel 1), the rotation (in 45° steps, 0–7), the scale (1–255), and the shape number (1 in this case). The shape beginning at location 23305 is always shape number 1. The next shape description would be for shape number 2, and so on.

Place 1 in location 23301 and execute the machine code routine to draw the shape.

Place 0 in location 23301 and execute the machine code routine to erase the shape.

The following program (8.1) allows experimentation with rotation and scale using the shape description devised for the square. This program may also be used as a simple test bed when developing any shape description, simply by replacing the given description with a new description for the shape under development.

Example 8.1

```
 10 REM SQUARE
 20 REM set colours
 30 PAPER 7: INK 0: BORDER 7
 40 CLS
100 REM set new ramtop
110 CLEAR 65015: REM 16K=32247
120 LET start=65016: REM 16K=32
248
130 LET s=start
140 LET length=352
150 FOR i=1 TO length
160 READ n
170 POKE s,n
180 LET s=s+1
190 NEXT i
200 REM place shape description
210 LET s$="PUPRPDPLE"
220 LET s=23305
```

Program 8.1 *(continues)*

```
230 FOR c=1 TO LEN s$
240 POKE s,CODE (s$(c))
250 LET s=s+1
260 NEXT c
300 REM fetch parameters
310 INPUT "scale ";scale
320 INPUT "rotation ";rotation
400 REM draw shape 1
410 LET x=126
420 LET y=88
430 POKE 23296,x
440 POKE 23297,y
450 POKE 23298,rotation
460 POKE 23299,scale
470 POKE 23300,1: REM shape 1
480 POKE 23301,1: REM plot
490 RANDOMIZE USR start
500 REM wait, then erase
510 PAUSE 100
520 POKE 23301,0
530 RANDOMIZE USR start
600 REM again ?
610 PRINT AT 0,0;"Another try ?
(y or n)"
620 LET k$=INKEY$
630 IF k$="" THEN  GO TO 620
640 IF k$<>"y" THEN  GO TO 700
650 CLS
660 IF k$="y" THEN  GO TO 300
700 REM finished
710 PAPER 7: INK 0: BORDER 7
720 CLS
1000 REM SHAPE HANDLER
1010 DATA 62,0,50,6,91,33,0,91,1
26,35
1020 DATA 70,17,6,0,25,119,120,3
5,119,33
1030 DATA 8,91,14,1,58,4,91,71,2
54,1
1040 DATA 40,11,35,126,254,69,32
,250,12,121
1050 DATA 184,32,245,229,245,241
,225,35,126,254
1060 DATA 80,40,8,254,78,32,93,5
5,63,24
1070 DATA 1,55,35,229,245,58,3,9
1,95,241
1080 DATA 213,245,48,9,33,7,91,7
8,35,70
1090 DATA 62,175,184,56,121,62,1
6,128,71,33
1100 DATA 0,64,254,128,48,9,17,0
,8,25
1110 DATA 254,64,48,1,25,203,184
,203,176,62
```



```
1120 DATA 63,144,17,32,0,254,8,5
6,5,214
1130 DATA 8,25,24,247,254,0,40,4
,61,36
1140 DATA 24,248,121,254,8,56,5,
214,8,35
1150 DATA 24,247,79,62,7,145,24,
8,24,151
1160 DATA 24,110,24,110,24,174,5
5,63,87,71
1170 DATA 94,254,0,40,4,203,11,1
6,252,203
1180 DATA 67,40,5,62,1,50,6,91,2
41,245
1190 DATA 48,13,58,5,91,254,1,32
,4,203
1200 DATA 195,24,2,203,131,122,6
6,254,0,40
1210 DATA 4,203,3,16,252,115,241
,209,225,229
1220 DATA 213,245,126,254,85,32,
4,22,1,24
1230 DATA 22,254,68,32,4,22,5,24
,14,254
1240 DATA 82,32,4,22,3,24,6,254,
76,32
1250 DATA 23,22,7,58,2,91,254,8,
48,14
1260 DATA 130,254,9,56,4,214,8,2
4,248,87
1270 DATA 24,8,24,87,24,79,24,14
2,24,134
1280 DATA 58,7,91,79,58,8,91,71,
122,254
1290 DATA 1,32,1,4,254,2,32,2,4,
12
1300 DATA 254,3,32,1,12,254,4,32
,2,12
1310 DATA 5,254,5,32,1,5,254,6,3
2,2
1320 DATA 5,13,254,7,32,1,13,254
,8,32
1330 DATA 2,13,4,121,50,7,91,120
,50,8
1340 DATA 91,241,209,245,29,123,
254,0,40,184
1350 DATA 241,213,245,24,177,241
,209,24,1,241
1360 DATA 225,201
```

Program 8.1

Notice the difference in the size of the square when rotations of 0, 2, 4, or 6 are used, compared to its size under rotations of 1, 3, 5, or 7.

This difference seems to appear even when the same scale factor is used with different values of rotation. Why is there a difference in size?

Think of the distance between two adjacent pixels, positioned as in Fig. 8.4. When measuring between pixels vertically or horizontally, the distance could be called 1 unit.

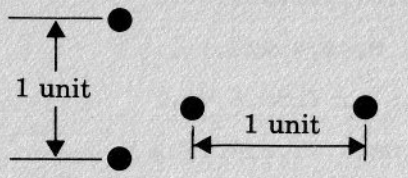


Figure 8.4 Pixels 1 unit apart

However, measuring at 45° to the horizontal, as in Fig. 8.5, the distance is approximately 1.4 units.

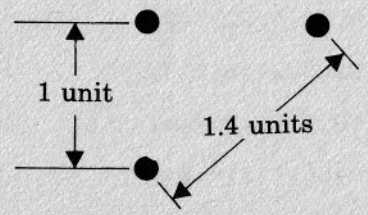


Figure 8.5 Comparison of vertical and inclined measurements

So, when a shape is rotated through 1, 3, 5, or 7 rotation units of 45° each, the resultant shape will appear 1.4 times as large. If this effect causes problems in any particular application, it can be overcome by using SCALE = 3 when ROTATION = 1, 3, 5, or 7 and SCALE =4 when ROTATION = 0, 2, 4, or 6.

In most cases, the difference in size is not a great problem.

RECTANGLE

The shape description for a rectangle can be constructed in a similar way to the description of the square. (See Fig. 8.6.)

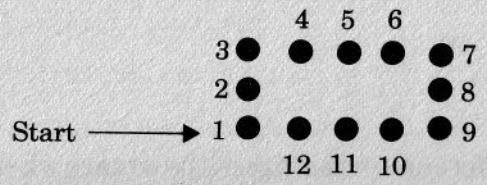


Figure 8.6 Pixels defining a rectangle

The codes for each pixel are:

Pixel 1: P U
2: P U
3: P R
4: P R
5: P R
6: P R
7: P D
8: P D
9: P L
10: P L
11: P L
12: P L

The shape description for the rectangle is:

PUPUPRPRPRPRPDPDPLPLPLPLE

Example 8.2

Use the Program given in Example 8.1, but replace the shape description by modifying line 210.

```
210 LET s$="PUPUPRPRPRPRPDPDPLPLPLPLE"
```

The starting point for the square and the rectangle was the bottom left corner in each case. This is not always convenient, because any rotation will take place about this point, effectively rotating the shape about its corner, which may make it a little difficult to predict where the shape will be drawn.

An alternative method is to define the shape in such a way that the origin is in the centre. If this is done, any rotations will take place about the centre, which will remain fixed. This means that some pixels are to be passed over without actually plotting them. The No plot instruction is used in this case. The rectangle in the last example could be re-defined as shown in Fig. 8.7.

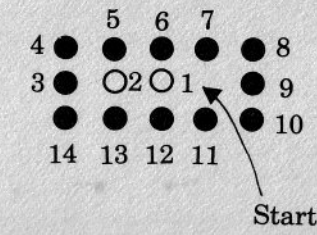


Figure 8.7 An alternative definition of the rectangle
(○ denotes a pixel which is Not to be plotted)

Pixel 1 is Not to be plotted, then move Left to pixel 2.
2 is Not to be plotted, then move Left to pixel 3.
3 is to be Plotted, then move Up to pixel 4.

4	P	R
5	P	R
6	P	R
7	P	R
8	P	D
9	P	D
10	P	L
11	P	L
12	P	L
13	P	L
14	P	U

The resulting shape description is:

```
210 LET s$="NLNLPUPRPRPRPRPDPDPLPLPLPLPUE"
```

Example 8.3

Test the shape description for the rectangle given above, using the program given in Example 8.1, changing only line 210.

Afterwards, make the following additional changes to the program.

```
 300 REM rotate and move
 310 FOR x=50 TO 200 STEP 50
 320 FOR y=30 TO 130 STEP 50
 330 FOR r=0 TO 7
 340 IF (r=0) OR (r=2) OR (r=4)
OR (r=6) THEN  LET scale=12
 350 IF (r=1) OR (r=3) OR (r=5)
OR (r=7) THEN  LET scale=9
 400 REM draw shape 1
 430 POKE 23296,x
 440 POKE 23297,y
 450 POKE 23298,r
 460 POKE 23299,scale
 470 POKE 23300,1: REM shape 1
 480 POKE 23301,1: REM plot
 490 RANDOMIZE USR start
 500 NEXT r
 510 NEXT y
 520 NEXT x
```

Program 8.3

Some shapes involve diagonal lines, as shown in Fig. 8.8, and these can be coded using a 'stepping' technique.

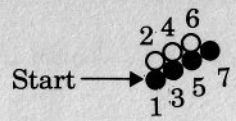


Figure 8.8 A diagonal line

Pixel 1: P U
2: N R
3: P U
4: N R
5: P U
6: N R
7: P U

Note that after Plotting pixel 7, some other direction may be more useful rather than Up, as shown, depending on the shape of the remainder of the object of which the diagonal line forms a part. This diagonal line is coded as:

```
"PUNRPUNRPUNRPUE"
```

Example 8.4

Use the program given in Example 8.1 to experiment with the diagonal line, by inserting the shape definition in line 210 as usual. Be prepared for some surprises when using scales larger than 1!

The diagonal line appears as expected when the scale is 1, but seems to change when the scale is greater than 1. Why is this? Consider the original definition, and the way in which the scale is implemented. Scaling is a repetitive operation in which the current TASK and MOVE instructions are repeated SCALE times.
So, if the instructions PU, shown in Fig. 8.9

Figure 8.9

are repeated (say) three times, the effect is as illustrated in Fig. 8.10, (i.e., like PUPUPU).

Figure 8.10 Repeating PU three times

However, if the sequence PUNR, shown in Fig. 8.11

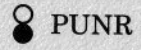


Figure 8.11

is repeated three times, the PU pair are first repeated three times, then the NR pair are repeated three times. The effect is shown in Fig. 8.12.

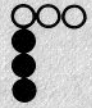


Figure 8.12 Repeating PUNR three times

The next TASK and MOVE pair will begin 3 pixels to the right of the last plotted pixel. This means that a step pattern for a diagonal line, as shown in Fig. 8.13, produces the pattern shown in Fig. 8.14, when a scale of three is used.

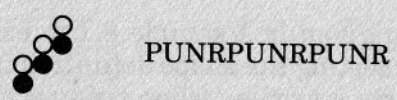


Figure 8.13 A basic step pattern

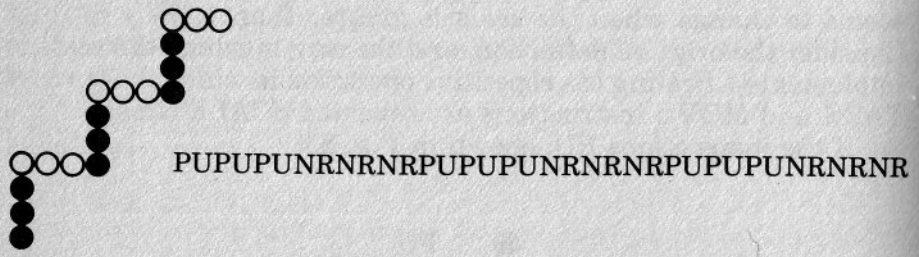


Figure 8.14 The step pattern, using a scale of three

This may or may not be desirable, but should be taken into account when designing a shape which is to be scaled.

The effects will be slightly different if the diagonal line does not lie at 45°, or if the line is produced using only P instructions, as shown in Fig. 8.15.

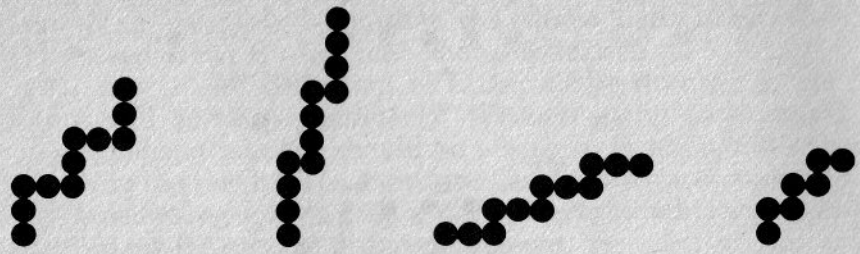


Figure 8.15 Diagonal lines at various angles

Many variations are possible, and the appearances are very acceptable, even when scaled.

Example 8.5

Using the program given in example 8.1, experiment with several different diagonal lines. The shape definitions should include some which are produced using only P TASK instructions.

Try using all possible rotation values with each shape definition.

LARGER SHAPES

Larger, more complex shapes can be defined in exactly the same way as the simple shapes considered so far. When a complex shape is being defined, it is often an advantage to go the 'long way' round all or part of the shape. This may be necessary to produce a shape definition which remains useful when scaled. In general, it is normally best to define shapes in as symmetrical a way as possible. Sometimes, this may involve moving over pixels already plotted (without plotting them a second time), or moving onto a pixel and then moving off it without plotting it.

Some of these tricks have been used to construct the shape definition for the lunar lander shown in Fig. 8.16.

Several of the pixels in the shape are to be used more than once, so each pixel's final state is shown. Follow the codes given, and trace the outline as it will be formed by the computer. Notice the extra codes to ensure that the figure is plotted symmetrically whatever scale is used.

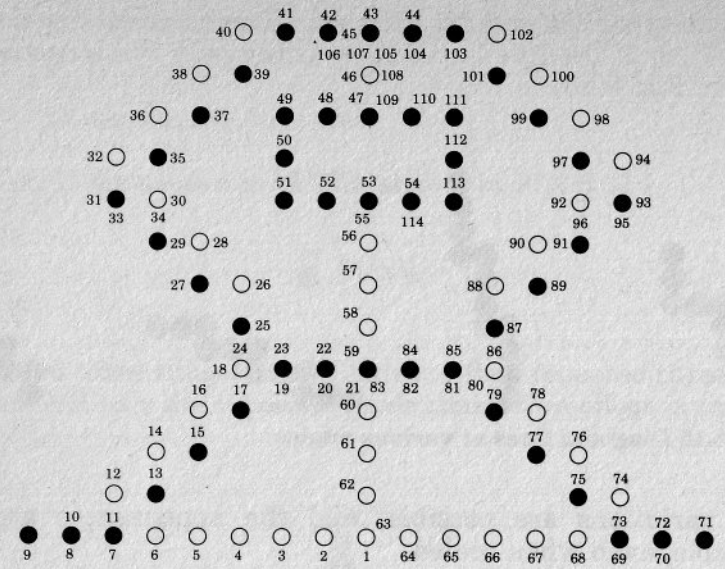


Figure 8.16 The outline of a lunar lander

● denotes a pixel which is to be plotted.
○ denotes a pixel which is not to be plotted.

```
 210 LET s$="NLNLNLNLNLNLNLNLPRP
RPUNRPUNRPUNRPUNRNRNRPLPLPLNUPUN
LPUNLPUNLPUNDNRNUPUNRPUNRPUNRPRP
RPRNLNDNDPLPLPDPDPRPRPRNLNDNDNDN
DNDNDNDNDNRNRNRNRNRNRNRNRPLPLPUN
LPUNLPUNLPUNLNLNLPRPRPRNUPUNRPUN
RPUNRPUNDNLNUPUNLPUNLPUNLPLPLPLN
RNDNDPRPRPDPDPLPLE"
```

Program 8.A

Example 8.6

Using the program given in Example 8.1, insert the shape definition given for the lunar lander, and try various scales and rotations. Then try improving the shape definition (which shouldn't be too difficult!). Try constructing a smaller shape definition, paying no attention to symmetry, and then see what happens when it is scaled.

MULTIPLE SHAPES

Additional shape descriptions are stored immediately after the first description, and can be stored at the same time, for example:


```
LET s$ ="PUPRPDPLEPUPUPRPRPRPRPDPDPLPLPLPLE"
```

lst shape — 2nd shape

The above defines two shapes, one after the other, and is placed in memory using the normal method, just as if one shape definition was being handled. Note that the shape definitions are *both* terminated using E after *each* shape definition.

The first shape definition is called shape number 1, and the second will be shape number 2. When plotting each shape, the required shape number should be placed in location 23300 before executing the machine code routine. In this way, many shapes can be handled using only one shape definition table which contains the descriptions for several different shapes.

The next example shows how to handle two different shapes held in the same table. It also illustrates the basic technique used to produce motion (although this technique will shortly be improved upon). As with character graphics, the most elementary technique for producing motion is:

(a) draw the shape.
(b) erase the shape.
(c) change the co-ordination of the starting point, and repeat from (a).

Some experimentation with delays between (a), (b), and (c) will produce different effects.

Example 8.7

```
  10 REM TWO SHAPES
  20 REM set colours
  30 PAPER 7: INK 0: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65015: REM 16K=32247
 120 LET start=65016: REM 16K=32
248
 130 LET s=start
 140 LET length=352
 150 FOR i=1 TO length
 160 READ n
 170 POKE s,n
 180 LET s=s+1
 190 NEXT i
 200 REM place shape description
 210 LET s$="NRNRNRNRNRNRNRNRNRN
RNRNRNRNUNLNLPLPLPLPLPLPLPLPLF
LNLNUNRPRPRPRPRPRPRPRPRPRPRPRP
RNUNLNLNLPLPLPLPLPLPLPLPLPLPLN
UNRPRPRPRPRPRPRPRPRNRNRNLNLNUNLN
```

Program 8.7 *(continues)*

```
LNLNLNLNLPLPLPLNUNRPRPRNUNLNLPLN
UNLNLENRNRNUNLPLPLNLNUNRPRPRNRNU
NLPLNLNUNRNRNRE"
 220 LET s=23305
 230 FOR c=1 TO LEN s$
 240 POKE s,CODE (s$(c))
 250 LET s=s+1
 260 NEXT c
 300 REM draw and erase shapes
 310 POKE 23298,1: REM rotation
 320 POKE 23299,1: REM scale
 330 LET x=10: REM initial x
 340 POKE 23297,100: REM y
 350 FOR i=1 TO 1000
 360 POKE 23296,x
 370 POKE 23301,1: REM plot
 380 POKE 23300,1: REM shape 1
 390 RANDOMIZE USR start: REM pl
ot shape 1
 400 POKE 23296,x-3: REM for sha
pe 2
 410 PRINT 23300,2: REM shape 2
 420 LET y=88
 430 POKE 23296,x
 440 POKE 23297,y
 450 POKE 23298,rotation
 460 POKE 23299,scale
 470 POKE 23300,2: REM shape 1
 480 RANDOMIZE USR start: REM pl
ot shape 2
 490 POKE 23301,0: REM erase
 500 PAUSE 5
 510 RANDOMIZE USR start: REM er
ase shape 2
 520 POKE 23300,1: REM shape 1
 530 RANDOMIZE USR start: REM er
ase shape 1
 540 LET x=x+1: REM next positio
n
 550 IF x>239 THEN  LET x=10: RE
M back to start, if shape would
be off the screen
 560 NEXT i
 700 REM finished
 710 PAPER 7: INK 0: BORDER 7
 720 CLS
```

Program 8.7

SOLID SHAPES

The use of the shape description system is not restricted solely to the outlines of shapes. It may also be used to handle 'solid' shapes which contain larger numbers of pixels grouped together, like the shape shown in Fig. 8.17.

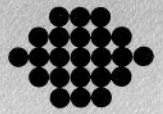


Figure 8.17 A solid shape

If the object is very small (8 × 8 or less), it might be better to treat it as a UDG character. However, if it is even slightly larger than a single character space it can be handled more quickly using the shape definition system. The disadvantage is that the shape definition table will normally occupy more space in memory than the corresponding UDG character definition. The advantages are that it will probably be plotted and erased more quickly; and rotation is possible in 45° steps instead of 90° steps.

Scaling a 'solid' shape will demand a special technique, otherwise a few 'holes' will appear in the shape. For instance, scaling the shape shown in Fig. 8.17, using a scale factor of 3, may produce the result shown in Fig. 8.18.

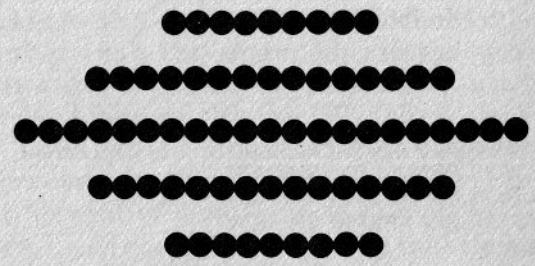


Figure 8.18 The solid shape of Figure 8.17, drawn using scale factor 3

However, the shape can be made to appear solid once more if it is plotted using a loop which plots the shape three times; once in the position shown (beginning at *x,y*) and at two other positions each beginning 1 pixel vertically up the screen from the last. Thus:

If the first shape begins at *x,y*
the second should begin at *x,y*+1
and the third should begin at *x,y*+2

After setting the *x* co-ordinate, the scale, rotation, and shape number, assign a value to *y* (the starting *y* co-ordinate), then use

```
FOR i = y TO y+2
POKE 23297,i
RANDOMIZE USR start
NEXT i
```

This plots the same shape three times, with the shapes interlaced so that the result is a 'solid' shape once again.

Example 8.8

Add lines 1000–1360 from example 8.1 (the SHAPE HANDLER routine) to Program 8.8.

This program plots the basic shape, then scales this shape using scale = 2, and then shows how to plot a solid shape when scale = 2, by plotting the shape twice. Finally, it plots the shape with scale = 3, and three times with scale = 3.

```
  10 REM SMUDGE
  20 REM set colours
  30 PAPER 7: INK 0: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65015: REM 16K=32247
 120 LET start=65016: REM 16K=32
248
 130 LET s=start
 140 LET length=352
 150 FOR i=1 TO length
 160 READ n
 170 POKE s,n
 180 LET s=s+1
 190 NEXT i
 200 REM place shape description
 210 LET s$="NUPLPRNUPLPLPRNRNUP
LPLPLPRNRNRNUPLPLPRNRNUPLPRNDNDN
DNDPRPLNUPRPRPLNLNUPRPRPRPLNLNLN
UPRPRPLNLNUPRPLE"
 220 LET s=23305
 230 FOR c=1 TO LEN s$
 240 POKE s,CODE (s$(c))
 250 LET s=s+1
 260 NEXT c
 300 REM plot shape
 310 POKE 23296,92: REM x
 320 POKE 23297,50: REM y
 330 POKE 23298,0: REM rotation
 340 POKE 23299,1: REM scale
 350 POKE 23300,1: REM shape no.
 360 POKE 23301,1: REM plot
 370 RANDOMIZE USR start
 380 PRINT AT 17,11;"1"
 400 REM plot, scale 2
 410 POKE 23299,2: REM scale
 420 POKE 23296,124: REM x
 430 RANDOMIZE USR start
 440 PRINT AT 17,15;"2"
 500 REM plot twice, scale 2
```



```
 510 POKE 23297,80: REM y
 520 RANDOMIZE USR start
 530 POKE 23297,81: REM y
 540 RANDOMIZE USR start
 600 REM plot scale 3
 610 POKE 23296,155: REM x
 620 POKE 23297,50: REM y
 630 POKE 23299,3: REM scale
 640 RANDOMIZE USR start
 650 REM plot 3 times, scale 3
 660 FOR y=80 TO 82
 670 POKE 23297,y
 680 RANDOMIZE USR start
 690 NEXT y
 700 PRINT AT 17,19;"3"
 800 REM end
```

Program 8.8

Smooth movement

The basic technique shown for producing movement with the shape definition system suffers from flicker. The same technique also produced flicker when used with the UDG characters, so it is the technique rather than the shape description system which is at fault. In that case, it should be possible to use the flicker-free technique evolved for use with the UDG characters. This made use of a row of blank pixels around the trailing edge of a moving character. However, the shape definition system does not have the capability at present of simultaneously plotting some pixels and erasing others. Until now, this has meant that the only way to erase pixels is to set the PLOT/ERASE flag in location 23301 to 0, and erase the entire shape. Fortunately, the shape handler can be modified very simply to allow some pixels to be plotted and others to be erased without changing the PLOT/ERASE flag. This means that any N TASK instruction will now be interpreted as 'erase' even when the shape is being plotted. A trailing group of pixels can now be incorporated in a shape definition using N TASK codes. Setting location 23301 to 1 will plot the shape and erase any pixels which have a TASK code of N. Setting location 23301 to 0 will erase the shape as before, removing the previously-plotted pixels while allowing any pixels erased using N codes to remain erased. It is now easy to

1. plot a shape;
2. calculate the next position for the shape;
3. plot the shape in the new position.

Step 3 automatically erases the previous shape (provided the

distance moved is not greater than the distance covered by trailing blanks). This technique removes the flicker, resulting in smooth movement, just as it did with the UDG characters.

The modifications to be made to the SHAPE HANDLER routine involve changing three of the DATA items. A completely new routine can be formed, and used on its own, or the original routine can be modified once it is loaded into memory. This second technique is useful, because it allows the basic routine to be held in memory and changed (or returned to its original state) very easily.

The following changes should be made to the SHAPE HANDLER DATA:

Change the 73rd item (currently 48) to 0.
Change the 74th item (currently 9) to 0.
Change the 182nd item (currently 13) to 11.

These changes can be made using

POKE start + 72,0
POKE start + 73,0
POKE start + 181,11

The routine which is produced allows overprinting of an object which includes 'trailing' blanks to erase the remains of the original shape as movement occurs.

The routine can be restored to normal using

POKE start + 72,48
POKE start + 73,9
POKE start + 181,13

Using both of the above sets of statements in a BASIC program allows both versions of the shape handler routine to be used when necessary, and provides maximum flexibility.

Example 8.9

Add the SHAPE HANDLER data to Program 8.9 (lines 1000–1360).

```
 10 REM MOVEMENT
 20 REM set colours
 30 PAPER 7: INK 0: BORDER 7
 40 CLS
100 REM set new ramtop
110 CLEAR 65015: REM 16K=32247
120 LET start=65016: REM 16K=32
248
130 LET s=start
140 LET length=352
150 FOR i=1 TO length
160 READ n
170 POKE s,n
180 LET s=s+1
190 NEXT i
200 REM place shape description
210 LET s$="NRNRNRNRNRNRNRNRNRN
RNRNRNRNUNLNLPLPLPLPLPLPLPLPLP
LNLNUNRPRPRPRPRPRPRPRPRPRPRPRP
RNUNLNLNLPLPLPLPLPLPLPLPLPLPLN
UNRPRPRPRPRPRPRPRPRNRNRNLNLNUNLN
LNLNLNLNLPLPLPLNUNRPRPRNUNLNLPLN
UNLNLE"
220 LET s=23305
230 FOR c=1 TO LEN s$
240 POKE s,CODE (s$(c))
250 LET s=s+1
260 NEXT c
300 REM modify SHAPE HANDLER
310 POKE start+72,0
320 POKE start+73,0
330 POKE start+181,11
400 REM move shape
410 POKE 23297,100: REM y
420 POKE 23298,0: REM rotation
430 POKE 23299,1: REM scale
440 POKE 23300,1: REM shape 1
450 POKE 23301,1: REM plot
460 FOR x=1 TO 240
470 POKE 23296,x
480 RANDOMIZE USR start
490 NEXT x
700 REM finished
710 PAPER 7: INK 0: BORDER 7
720 CLS
```

Program 8.9

Lines 310, 320, and 330 modify the SHAPE HANDLER routine so that it is capable of handling plotting and unplotting within the same shape description, to allow smooth movement.

The collision flag

When the SHAPE HANDLER routine begins, the collision flag in location 23302 is reset to 0 automatically. As the shape is plotted, each pixel position at which plotting takes place is examined. If any of these positions already contain a lit pixel, the collision flag in location 23302 is set to 1.

So, when the routine has finished, and control returns to BASIC,

the contents of location 23302 will indicate whether the shape collided with one or more lit pixels when it was drawn.

IF PEEK 23302 = 1 THEN a collision took place.
IF PEEK 23302 = 0 THEN there was no collision.

Example 8.10

Two UDG characters are used to represent a spacecraft and an alien. Press key 'f' to shoot the alien, and examine the collision flag in action.

```
  10 REM ZAP
  20 REM set colours
  30 PAPER 7: INK 0: BORDER 7
  40 CLS
 100 REM set new ramtop
 110 CLEAR 65015: REM 16K=32247
 120 LET start=65016: REM 16K=32
248
 130 LET s=start
 140 LET length=352
 150 FOR i=1 TO length
 160 READ n
 170 POKE s,n
 180 LET s=s+1
 190 NEXT i
 200 REM place shape description
 210 LET s$="PRNRNRNRPRPRPRE"
 220 LET s=23305
 230 FOR c=1 TO LEN s$
 240 POKE s,CODE (s$(c))
 250 LET s=s+1
 260 NEXT c
 300 REM define UDG characters
 310 FOR i=1 TO 3
 320 READ c$
 330 FOR j=0 TO 7
 340 READ d
 350 POKE USR c$+j,d
 360 NEXT j
 370 NEXT i
 400 REM place UDG characters
 410 PRINT AT 9,0;CHR$ 144+CHR$
145
 420 PRINT AT 9,31;CHR$ 146
 500 REM move shape
 510 POKE 23297,97: REM y
 520 POKE 23298,0: REM rotation
 530 POKE 23299,1: REM scale
 540 POKE 23300,1: REM shape 1
 550 PRINT AT 21,0;"Press  f  to
 fire."
```



```
 560 LET k$=INKEY$
 570 IF k$="" THEN  GO TO 560
 580 REM fire laser
 590 LET x=20
 600 POKE 23296,x
 610 POKE 23301,1: REM plot
 620 RANDOMIZE USR start
 630 IF PEEK 23302=1 THEN  GO TO
 700
 640 POKE 23301,0: REM erase
 650 RANDOMIZE USR start
 660 LET x=x+3
 670 IF x<=247 THEN  GO TO 600
 700 REM remove alien
 710 PRINT AT 9,28;"ZAP!"
 800 REM finished
1400 REM UDG DEFINITIONS
1410 DATA "A",96,96,112,127,127,
127,0,0
1420 DATA "B",0,0,0,192,240,254,
0,0
1430 DATA "C",0,0,24,60,126,60,2
4,36
```

Program 8.10

The laser beam fired from the spacecraft is created using the shape definition illustrated in Fig. 8.19.

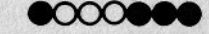


Figure 8.19 The laser beam

This shape is handled by repeating the following steps:
(a) check the shape will be on the screen.
(b) plot the shape
(c) examine the collision flag, exiting if it is 1
(d) erase the shape.
When a collision is detected, the alien is removed from the screen. Try running the program again, after making the following change:

```
420 PRINT AT 10,31;CHR$ 146
```

What happens to the alien? Why?

Remember to add the SHAPE HANDLER data to the program (lines 1000–1360) before running it.

Larger shape description tables

Occasionally, there is a need for larger shape description tables. This is particularly true when multiple shapes are being handled. The current SHAPE HANDLER routine demands that the shape description table should start at location 23305. Since the printer buffer ends at location 23551, this limits the length of the table to 247 instructions. However, it is easy to modify the SHAPE HANDLER routine to allow the start of the table to be set at any convenient location. This allows a larger table to be accommodated, usually by holding it above RAMTOP, where it can be very large provided RAMTOP is set suitably.

The modifications to be made to SHAPE HANDLER are made from BASIC. If 'table' denotes the start address of the table; and if 'start' denotes the start of the SHAPE HANDLER routine:

```
POKE start+20, table-256* INT (table/256)
POKE start+21, INT (table/256)
```

Normally, the SHAPE HANDLER routine would be loaded from data in the usual way, and then the modifications would be made.

This modification also allows the use of multiple tables simply by redefining 'table' and using the two POKE statements whenever required. However, since a shape definition table may be of any length, this facility is not normally needed.

9 SATURN'S SURFACE

At last! After years of travelling through the dark and lonely wastes of the solar system, Saturn's surface fills the window of the flight deck. The little craft has made it!

But wait – the onboard computers have failed on the way through a dense belt of radiation which encircles the planet (the engineers should have installed some Spectrums instead!) and the ship hurtles towards the planet's surface, out of control. Save this mission by guiding the spacecraft to a soft landing on the low, flat area on Saturn's surface. Oh, by the way, there is only a little fuel left in the tank. . .

Use keys 5, 7, and 8 to move the ship left, up, and right respectively.

Like DOMINOES, SATURN'S SURFACE is a large program; too large to fit the 16K Spectrum as one complete program. However, it can be made to fit by breaking it into sections and loading the sections separately. This technique is used by many commercial software houses, because it allows the program to be stored efficiently.

The machine code routines can be stored as blocks of code, which means that it is not necessary to hold the DATA to construct the routines; nor is it necessary to incorporate a loading loop in the program. Dispensing with this DATA saves a substantial amount of space.

UDG character definitions can also be loaded as blocks of code, which further reduces the length of the program.

An essential feature of the program is a view of the planet's surface, much of which is constructed from DATA. This picture will be constructed from a set of heights held in DATA statements in the main program. Although this increases the length of the main program, it is important that the planet's surface can be constructed from within this program so that the game may be repeated without having to load a screen separately from tape each time.

SATURN'S SURFACE consists of the following components:

1. A main program which loads (2) and (3) before the game begins.
2. UDG character definitions, constructed using a separate program, and stored as CODE.

3. Machine code routines for scrolling and character plotting, constructed using a separate program, and saved as CODE.

The programs should be saved on tape in the order shown above.

UDG character definitions

The UDG characters required for SATURN'S SURFACE are constructed using Program 9.A.

```
  10 REM UDG DEFINITIONS
 100 FOR i=1 TO 21
 110 READ c$
 120 FOR j=0 TO 7
 130 READ d
 140 POKE USR c$+j,d
 150 NEXT j
 160 NEXT i
 200 REM UDG CODES
 210 DATA "A",0,24,126,126,60,36
,66,0
 220 DATA "B",0,0,0,12,62,127,25
4,255
 230 DATA "C",31,63,127,127,255,
127,127,28
 240 DATA "D",248,254,254,255,25
5,254,120,32
 250 DATA "E",24,62,126,63,126,2
54,56,24
 260 DATA "F",0,0,0,0,0,0,2,1
 270 DATA "G",0,0,0,0,0,0,64,64
 280 DATA "H",1,4,0,0,0,0,0,0
 290 DATA "I",0,64,0,0,0,0,0,0
 300 DATA "J",0,0,0,0,4,2,7,3
 310 DATA "K",0,0,0,0,64,16,192,
160
 320 DATA "L",1,4,0,0,0,0,0,0
 330 DATA "M",208,128,8,0,0,0,0,
0
 340 DATA "N",0,0,0,16,0,37,15,7
 350 DATA "O",0,8,0,0,36,64,128,
192
 360 DATA "P",15,37,10,0,32,4,0,
0
 370 DATA "Q",192,128,232,0,48,6
4,0,0
 380 DATA "R",1,146,0,73,15,47,3
1,143
 390 DATA "S",0,1,128,228,192,22
6,72,224
 400 DATA "T",7,21,98,134,32,64,
12,128
 410 DATA "U",68,96,50,16,16,68,
0,17
```

Program 9.A

Run this program, to place the definitions in memory, then save these as the *second* program on the tape, using

(48K)

SAVE "udg" CODE 65368,168

(16K)

SAVE "udg" CODE 32600,168

The routines

The three machine code routines are:

DISPLAY: FAST BIT LEFT (WRAP)
DISPLAY: FAST BIT RIGHT (WRAP)
CHARACTER PLOTTER

These routines are placed in memory using Program 9.B.

```
  10 REM CONSTRUCT ROUTINES
  20 REM set new ramtop
  30 CLEAR 65096: REM 16K=32328
 100 REM LOAD ROUTINES
 110 LET s=65097: REM 16K=32329
 120 LET length=34+34+203
 130 FOR i=1 TO length
 140 READ n
 150 POKE s,n
 160 LET s=s+1
 170 NEXT i
1000 REM DISPLAY  FAST BIT LEFT
(WRAP)
1010 DATA 33,0,64,6,192,197,6,31
,94,203
1020 DATA 19,245,35,94,203,19,43
,94,203,19
1030 DATA 115,35,16,244,241,94,2
03,19,115,35
1040 DATA 193,16,228,201
1100 REM DISPLAY  FAST BIT RIGHT
 (WRAP)
1110 DATA 33,255,87,6,192,197,6,
31,94,203
1120 DATA 27,245,43,94,203,27,35
,94,203,27
1130 DATA 115,43,16,244,241,94,2
03,27,115,43
1140 DATA 193,16,228,201
1200 REM CHARACTER PLOTTER
1210 DATA 58,2,91,254,32,56,76,2
54,128,56
1220 DATA 10,254,144,56,68,254,1
65,56,9,24
```

Program 9.B *(continues)*

```
1230 DATA 62,33,7,61,214,32,24,6
,42,123
1240 DATA 92,43,214,143,17,8,0,2
54,0,40
1250 DATA 4,25,61,24,248,6,0,126
,43,229
1260 DATA 14,0,24,1,241,7,245,19
7,245,33
1270 DATA 0,91,126,254,249,48,18
,129,79,35
1280 DATA 126,254,169,48,10,128,
71,24,8,24
1290 DATA 229,24,220,24,117,24,1
11,62,16,128
1300 DATA 71,33,0,64,254,128,48,
9,17,0
1310 DATA 8,25,254,64,48,1,25,20
3,184,203
1320 DATA 176,62,63,144,17,32,0,
254,8,56
1330 DATA 5,214,8,25,24,247,254,
0,40,4
1340 DATA 61,36,24,248,121,254,8
,56,5,214
1350 DATA 8,35,24,247,79,62,7,14
5,55,63
1360 DATA 87,71,94,254,0,40,4,20
3,11,16
1370 DATA 252,241,56,4,203,131,2
4,2,203,195
1380 DATA 122,66,254,0,40,4,203,
3,16,252
1390 DATA 115,193,12,121,254,8,3
2,147,4,241
1400 DATA 225,120,254,8,32,141,4
0,4,241,193
1410 DATA 241,225,201
```

Program 9.B

Run this program, then save the routines as the *third* program on the tape, using

(48K)

SAVE "routines" CODE 65097,271

(16K)

SAVE "routines" CODE 32328,271

The main program

Program 9.C should be entered and saved as the *first* program on the

tape (after making the changes required in lines 30, 60, and 70, if a 16K machine is being used).

```
  10 REM SATURN'S SURFACE
  20 REM load components
  30 CLEAR 65096: REM 16K=32329
  40 PAPER 7: INK 7: CLS : BORDE
R 7
  50 PRINT  INK 0;AT 10,11;"LOAD
ING..."
  60  PRINT AT 19,0: LOAD "udg"C
ODE 65368,168: REM 16K=32600,168
  70 PRINT AT 19,0: LOAD "routin
es"CODE 65097,271: REM 16K=32330
,271
  80 CLS : GO SUB 4000: REM DRAW
 LANDSCAPE
  90 PAPER 6: INK 2: BORDER 1
 100 REM START OF ROUTINES
 110 LET bitleft=65097: REM 16K=
32329
 120 LET bitright=bitleft+34
 130 LET plot=bitright+34
 200 REM INITIALISE VARIABLES
 210 LET xc=23296: REM x co-ord
 220 LET yc=23297: REM y co-ord
 230 LET cc=23298: REM character
 code
 240 LET x=64: REM start
 250 LET y=160: REM start
 260 LET f=100: REM fuel
 300 REM DRAW LARGE ASTEROIDS
 310 FOR c=16 TO 208 STEP 32
 320 POKE xc,c+4: POKE yc,104: P
OKE cc,145
 330 RANDOMIZE USR plot
 340 PRINT AT 9,c/8;CHR$ 146+CHR
$ 147
 350 NEXT c
 400 REM PRINT SMALL ASTEROIDS
 410 PRINT AT 11,5;CHR$ 148;AT 1
1,9;CHR$ 148;AT 11,14;CHR$ 148;A
T 11,18;CHR$ 148;AT 11,23;CHR$ 1
48;AT 11,27;CHR$ 148;AT 12,7;CHR
$ 148;AT 12,13;CHR$ 148;AT 12,24
;CHR$ 148;AT 16,26;CHR$ 148;AT 1
3,25;CHR$ 148;AT 12,29;CHR$ 148
 500 REM PLACE SHIP INITIALLY
 510 POKE cc,144: POKE xc,x: POK
E yc,y
 520 RANDOMIZE USR plot
1000 REM MAIN CONTROL LOOP
1010 POKE yc,y: RANDOMIZE USR pl
ot
```

Program 9.C *(continues)*

```
1020 LET k$=INKEY$
1030 IF k$="5" AND f>0 THEN  RAN
DOMIZE USR bitright
1040 IF k$="8" AND f>0 THEN  RAN
DOMIZE USR bitleft
1050 IF k$="7" THEN  LET y=y+2:
LET f=f-1: IF f<=0 THEN  PRINT
INVERSE 1;AT 21,0;"              o
ut of fuel!          ": LET y=y-2
1060 LET y=y-1: IF y>168 THEN  L
ET y=168
1070 IF POINT (x,y)=1 OR POINT (
x+7,y)=1 OR POINT (x+7,y+7)=1 OR
 POINT (x,y+7)=1 THEN  GO TO 300
0
1080 GO TO 1010
2000 REM SOFT LANDING
2010 POKE yc,y: RANDOMIZE USR pl
ot
2020 PRINT  INK 0;AT 2,3;"WELL D
ONE! A SOFT LANDING!"
2030 PRINT  INK 0;AT 4,1;"ANOTHE
R GIANT STEP FOR MANKIND."
2040 RETURN
2500 REM CRASHED
2510 LET l=(176-y)/8
2520 LET c=8
2530 POKE xc,x: POKE yc,y: POKE
cc,32: RANDOMIZE USR plot
2540 REM EXPLOSION EXPANDING
2550 FOR j=0 TO 3
2560 PRINT  OVER 1;AT l-1,c;CHR$
 (149+4*j)+CHR$ (150+4*j)
2570 PRINT  OVER 1;AT l,c;CHR$ (
151+4*j)+CHR$ (152+4*j)
2580 PAUSE 3
2590 NEXT j
2600 REM EXPLOSION CONTRACTING
2610 FOR j=3 TO 0 STEP -1
2620 PRINT  OVER 1;AT l-1,c;CHR$
 (149+4*j)+CHR$ (150+4*j)
2630 PRINT  OVER 1;AT l,c;CHR$ (
151+4*j)+CHR$ (152+4*j)
2640 PAUSE 3
2650 NEXT j
2660 PRINT  INK 0;AT 2,3;"SPLAT!
 YOU BLEW IT!"
2670 PRINT  INK 0;AT 4,3;"ANOTHE
R HUGE CRATER!"
2680 RETURN
3000 REM COLLISION
3010 IF y=10 AND f>=0 THEN  GO S
UB 2000: REM soft landing
3020 IF y<>10 OR f<0 THEN  GO SU
B 2500: REM crashed
```

Program 9.C *(continues)*


```
3500 REM ANOTHER GAME ?
3510 FOR w=1 TO 800: NEXT w
3520 CLS : PRINT AT 0,0;"Another
 game (y  or  n)?"
3530 LET k$=INKEY$: IF k$="" THE
N  GO TO 3530
3540 IF k$="y" THEN  GO TO 80
3550 GO TO 5000: REM end
4000 REM CONSTRUCT LANDSCAPE
4010 PAPER 6: INK 2: BORDER 1: C
LS
4020 RESTORE 4100
4030 FOR i=0 TO 255
4040 READ h
4050 PLOT i,0
4060 DRAW 0,h
4070 NEXT i
4080 RETURN
4100 REM LANDSCAPE HEIGHTS
4110 DATA 70,69,67,63,61,57,55,5
2,50,47
4120 DATA 45,42,39,37,35,32,29,2
7,25,22
4130 DATA 22,24,25,27,30,31,33,3
5,37,40
4140 DATA 39,37,35,33,31,29,27,2
5,24,22
4150 DATA 22,25,29,34,38,41,46,4
9,54,58
4160 DATA 60,57,55,53,51,49,47,4
6,44,42
4170 DATA 39,37,35,33,31,29,27,2
5,23,21
4180 DATA 21,24,27,30,33,37,40,4
3,46,50
4190 DATA 49,47,45,43,41,39,37,3
6,34,32
4200 DATA 32,34,34,37,38,39,40,4
1,43,44
4210 DATA 46,48,49,50,52,53,55,5
6,57,59
4220 DATA 58,54,50,45,41,37,33,2
9,25,21
4230 DATA 21,21,22,22,22,23,24,2
4,25,25
4240 DATA 26,26,27,27,28,28,29,2
9,30,30
4250 DATA 31,32,34,35,37,39,40,4
1,43,44
4260 DATA 45,47,48,50,52,53,54,5
6,57,59
4270 DATA 60,57,55,53,51,48,47,4
5,43,41
4280 DATA 41,45,48,51,54,57,60,6
3,67,70
```

```
4290 DATA 66,62,57,50,44,38,31,2
7,20,14
4300 DATA 10,10,10,10,10,10,10,1
0,10,10
4310 DATA 10,10,10,10,10,10,10,1
0,10,10
4320 DATA 11,14,18,21,24,28,30,3
3,36,39
4330 DATA 40,38,35,33,31,29,27,2
5,23,21
4340 DATA 21,25,30,34,38,41,45,5
0,55,60
4350 DATA 59,56,54,54,51,49,46,4
6,43,41
4360 DATA 42,47,50,55,63,68
5000 REM END OF GAME
5010 PAPER 7: INK 0: BORDER 7: C
LS
```

Program 9.C

When the program is run, remember to let the recorder play until 'udg' and 'routines' have been loaded. When the planet's surface begins to appear, the recorder can be stopped.

Program notes

10–90		Initialize
	30	Set RAMTOP.
	40	Set INK and PAPER to white.
	50	Print 'LOADING. . .' in black INK on a white background.
	60	Load the UDG character definitions. The normal loading message is printed 'invisibly' in white INK on white PAPER.
	70	Load the machine code routines, printing the loading message invisibly.
	80	Draw the landscape, using the subroutine at 4000.
	90	Set PAPER to yellow, and INK to red.
100–130		The start address of each machine code routine is defined.
200–260		Variables are defined.
300–350		A number of large asteroids are drawn, using the CHARACTER PLOTTER routine as well as ordinary PRINT statements.
400–410		Several small asteroids are positioned using PRINT statements.
500–520		The ship is drawn in its starting position.
1000–1080		The action in the game is handled using a loop
	1010	Plot the ship at the current position.
	1020	Fetch a keypress.
	1030	Move the screen 1 pixel to the right, if required.
	1040	Move the screen 1 pixel to the left, if required.
	1050	If the ship is to move upwards, add 2 to the current y co-ordinate and subtract 1 from the remaining fuel – but if there is no fuel left, print a message and ignore the upward move by restoring y to the original value.
	1060	Subtract 1 from the value of y, to represent gravity. If y is too great (i.e., $y>168$, which would plot the ship off the screen) restrict it to a maximum of 168.
	1070	Examine the four corners of the new position of the ship. If there is a lit pixel at any corner, assume the ship has collided with something, and exit via 3000.
	1080	If there are no lit pixels in the corners of the new position, loop to 1010 and use the new position as the current position.
2000–2040		This subroutine is used when the ship has landed successfully.
2500–2660		This subroutine is used when a collision is detected between the ship and any pixel other than those at the landing site.
	2510–2520	Find the line and column co-ordinates of the PRINT position nearest the current position of the ship.
	2530	Erase the ship by plotting a space character (code 32).
	2540–2680	Print UDG characters representing an explosion expanding and contracting.
3000–3550		Continue here if a lit pixel is detected at any corner of the ship's new position.
	3010	If $y = 10$ then the ship has landed, since the landing area is the lowest section of the landscape and is the only place which has y

co-ordinate 10. The ship cannot reach any other place with y co-ordinate 10 without first colliding with another part of the landscape.

3020 If $y<>10$ then the ship has crashed.

3500–3550 Either restart at line 80, or end the game via line 5000.

4000–4360 Construct the landscape.

4010 Ensure that PAPER is yellow, and INK is red.

4020 Set the DATA pointer to the beginning of the list of heights. This is not essential when the list is used for the first time, but it is required when the game is to be run again.

4030–4070 Read each height, then plot on the x axis (along the bottom of the screen) and draw a vertical line of the required height.

4080 End of subroutine.

4100–4360 DATA containing the heights of the points on the surface of the planet. These heights are stored in the order in which they will be used to construct the surface, from left to right, on the screen.

5000 End.

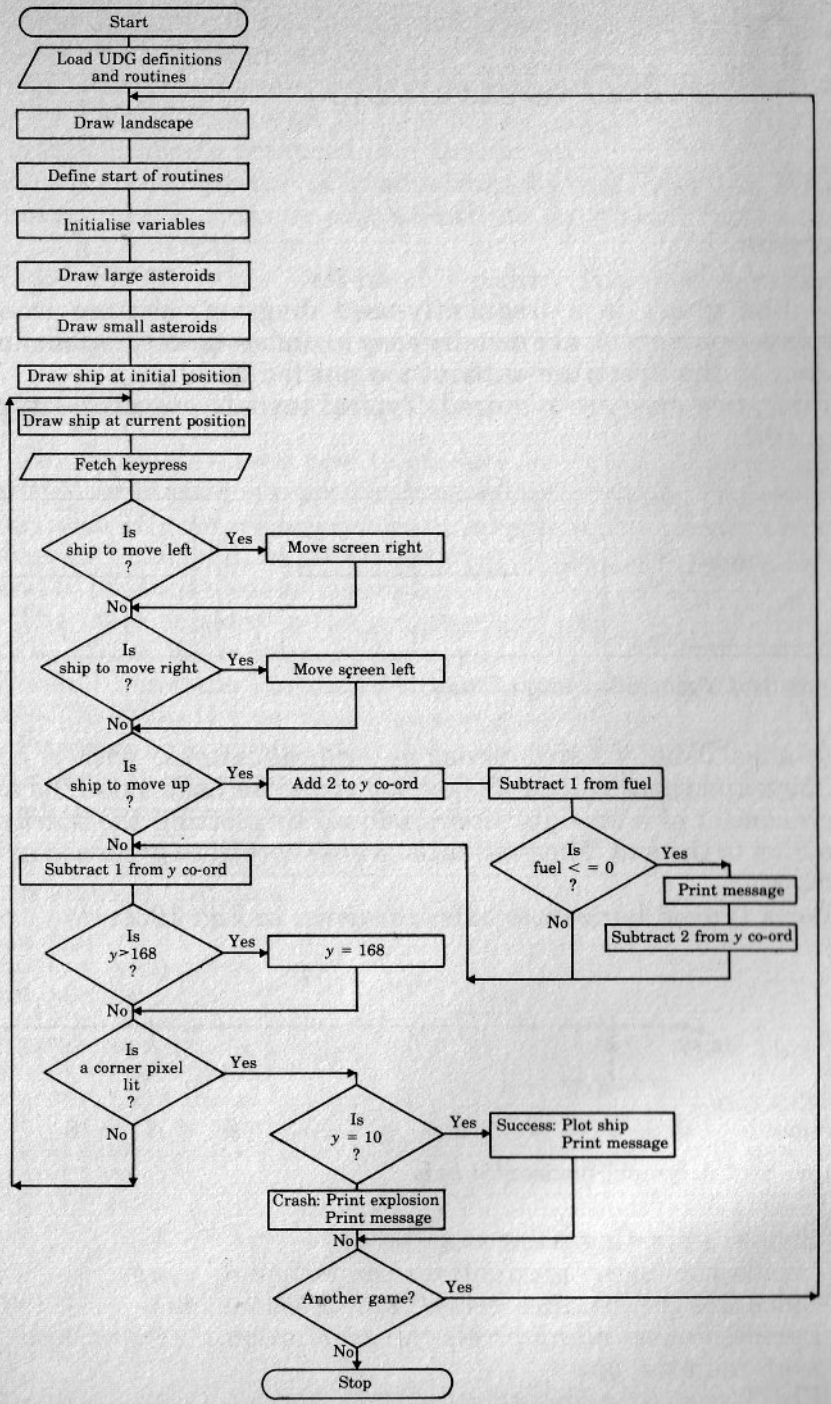


Figure 9.1 Saturn's Surface

10 GRAPHS AND CHARTS

Graphs

The line graph is a frequently-used diagram, because results displayed on a graph are usually easy to understand. Graphs can be drawn on the Spectrum without too much difficulty.

First, two axes are required. Typical layouts are illustrated in Fig. 10.1.

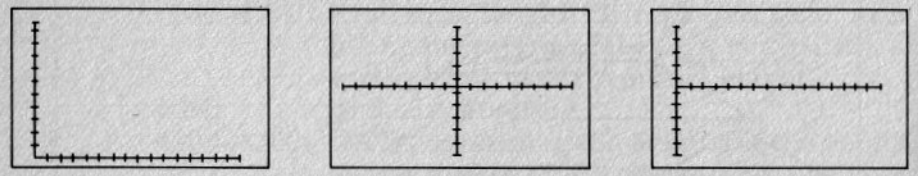


Figure 10.1 Typical layouts of axes

The axes usually carry divisions or graduations, with a scale nearby, and there will be a legend or name for each axis. The axes each consist of a straight line produced by plotting the start and drawing to the end. The graduations are short lines produced in the same way.

For a typical horizontal axis (as shown in Fig. 10.2):

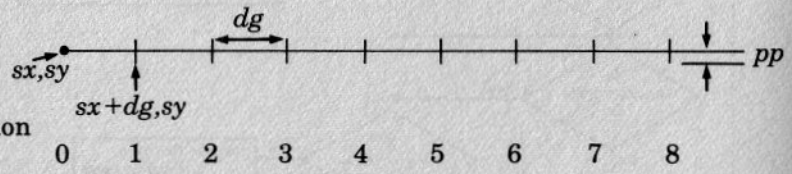


Figure 10.2 A typical horizontal axis

1. Choose a starting point *sx,sy*
2. Decide how many graduations there should be, *ng*
3. Calculate the distance between graduations, *dg*
4. Decide by how many pixels the graduation marks will project from the axis, *pp*
5. PLOT *sx,sy* (the first point on the axis)

6. DRAW *ng*dg*,0 (to the end of the axis)
7. For each of the graduations, from 0 to *ng*,
   (a) PLOT the bottom point (*sx* + graduation number**dg*, *sy* −*pp*)
   (b) DRAW 0,*pp* + *pp*

Vertical axes are produced in a similar way.

Denoting the number of graduations by *yng*, and the distance between graduations by *ydg*, substitute for (6) and (7) as follows:

(6) DRAW 0, *yng*ydg*

(7) (a) PLOT the left-hand point (*sx* −*pp*,*sy* +graduation number**ydg*)
   (b) DRAW *pp* +*pp*,0

Example 10.1

This example shows how to produce two axes; a horizontal axis with 12 graduations spaced at intervals of 15 pixels, and a vertical axis with 10 graduations spaced at intervals of 10 pixels. The short program (10.1) will be developed over the next few pages, to show how to produce various diagrams.

The main sections in the program are:

10– 40 Set the colours for the axes.
300–460 Draw the horizontal axis and graduations
600–730 Draw the vertical axis and graduations.

Program 10.1 is entitled 'BANK BALANCE GRAPH', because it will show the variation in the monthly bank balance for a small computer shop, when it is finished.

```
 10 REM BANK BALANCE GRAPH
 20 REM set colours
 30 PAPER 7: INK 0: BORDER 7
 40 CLS
300 REM HORIZONTAL AXIS
310 REM START AT 33,33
320 REM 12 GRADUATIONS
330 REM SPACED 15 APART
340 REM PROJECTING 2
350 REM initialise variables
360 LET sx=33: LET sy=33
370 LET ng=12
380 LET dg=15
390 LET pp=2
400 REM draw horizontal
410 PLOT sx,sy: DRAW ng*dg,0
420 REM add graduations
430 FOR n=0 TO ng
440 PLOT sx+n*dg,sy-pp
450 DRAW 0,pp+pp
460 NEXT n
600 REM VERTICAL AXIS
```

Program 10.1 *(continues)*

```
610 REM START AT 30,30
620 REM 10 GRADUATIONS
630 REM SPACED 10 APART
640 REM initialise variables
650 LET yng=10
660 LET ydg=10
670 REM draw vertical
680 PLOT sx,sy: DRAW 0,yng*ydg
690 REM add graduations
700 FOR n=0 TO yng
710 PLOT sx-pp,sy+n*ydg
720 DRAW pp+pp,0
730 NEXT n
```

Program 10.1

A scale can be added to an axis using standard 8 × 8 characters. To achieve precise control over the positioning of the characters, use the CHARACTER PLOTTER routine detailed in Chapter 1.

Before using the routine, it is essential that the position of the bottom left corner of the character block is known. Figure 10.3 shows a character block, and its position is calculated in much the same way as for the graduations.

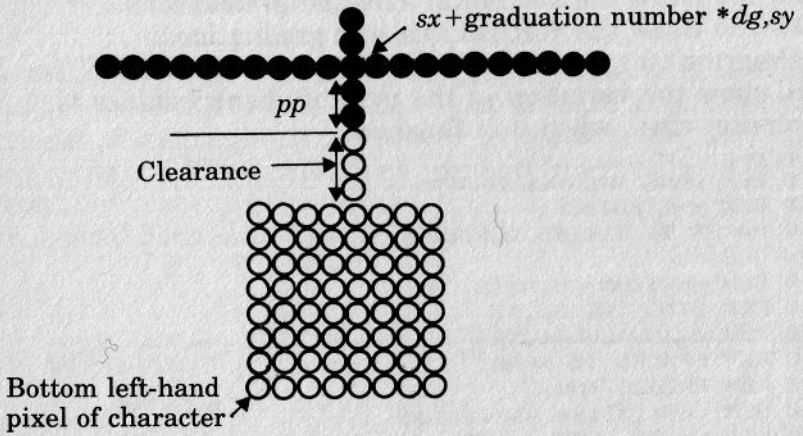


Figure 10.3 Position of a character block in relation to a graduation

The co-ordinates of the bottom left corner of the character block will be:

$sx-4+dg*$graduation number$,sy-5-8$

Figure 10.4 shows that some adjustment of the position will be required if a graduation is to be accompanied by more than one character.

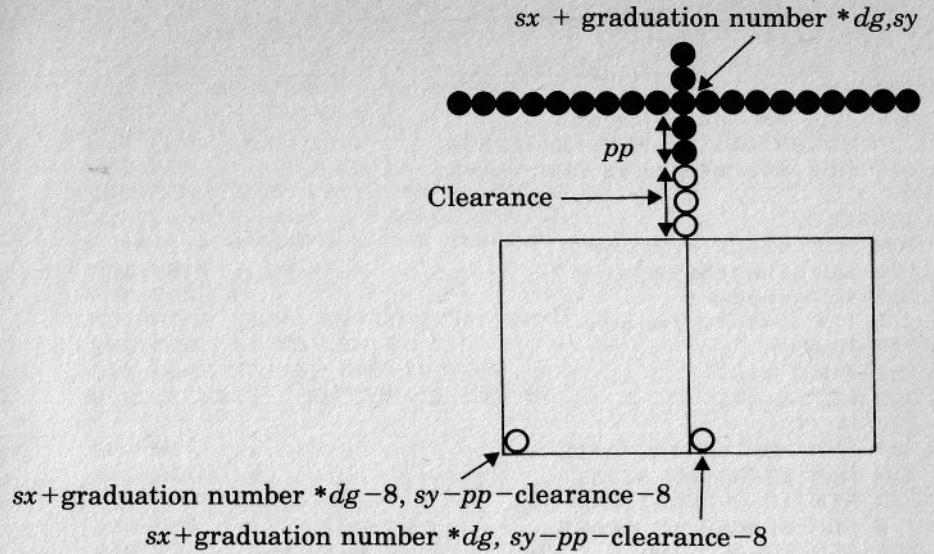


Figure 10.4 Two characters beside a graduation

Adding a label to an axis can be done in much the same way as adding a scale. However, there is no need to place characters next to graduation marks.

For a horizontal axis:

1. decide where to begin the label.
2. For each character,
   (a) use the CHARACTER PLOTTER routine to produce the character.
   (b) Add 8 to the x co-ordinate to find the position of the next character.

For a vertical axis, 2.(b) is replaced by

2. (b) Subtract 8 from the y co-ordinate to find the position of the next character.

This will result in a vertical label, with the characters one underneath the other, read downwards.

A vertical label may have its characters turned through 90° if required, by using the ROTATED SCALED CHARACTER PLOTTER routine, with a scale of 1 and a rotation of 1.

Example 10.2

Building on the last example, the axes are labelled.

```
 10 REM BANK BALANCE GRAPH
 20 REM set colours
 30 PAPER 7: INK 0: BORDER 7
 40 CLS
100 REM set new ramtop
110 CLEAR 65164: REM 16K=32396
120 LET start=65165: REM 16K=32
397
130 CLS
200 REM place routine
210 LET length=203
220 LET s=start
230 FOR i=1 TO length
240 READ n
250 POKE s,n
260 LET s=s+1
270 NEXT i
300 REM HORIZONTAL AXIS
310 REM START AT 33,33
320 REM 12 GRADUATIONS
330 REM SPACED 15 APART
340 REM PROJECTING 2
350 REM initialise variables
360 LET sx=33: LET sy=33
370 LET ng=12
380 LET dg=15
390 LET pp=2
400 REM draw horizontal
410 PLOT sx,sy: DRAW ng*dg,0
420 REM add graduations
430 FOR n=0 TO ng
440 PLOT sx+n*dg,sy-pp
450 DRAW 0,pp+pp
460 NEXT n
500 REM plot horizontal labels
510 FOR i=1 TO ng
520 READ l$
530 FOR j=1 TO LEN l$
540 POKE 23296,sx-4+dg*i
550 POKE 23297,sy-5-8*j
560 POKE 23298,CODE l$(j)
570 RANDOMIZE USR start
580 NEXT j
590 NEXT i
600 REM VERTICAL AXIS
610 REM START AT 30,30
620 REM 10 GRADUATIONS
630 REM SPACED 10 APART
640 REM initialise variables
650 LET yng=10
660 LET ydg=10
670 REM draw vertical
680 PLOT sx,sy: DRAW 0,yng*ydg
690 REM add graduations
700 FOR n=0 TO yng
```

Program 10.2 *(continues)*


```
710 PLOT sx-pp,sy+n*ydg
720 DRAW pp+pp,0
730 NEXT n
800 REM plot vertical labels
810 FOR i=1 TO yng+1
820 READ l$
830 FOR j=1 TO LEN l$
840 POKE 23296,sx-29+8*j: POKE
23297,sy-12+ydg*i: POKE 23298,CO
DE l$(j): RANDOMIZE USR start
850 NEXT j: NEXT i
860 READ l$
870 POKE 23297,sy+(yng+1)*ydg+5
880 FOR j=1 TO LEN l$
890 POKE 23296,-8+8*j: POKE 232
98,CODE l$(j): RANDOMIZE USR sta
rt
900 NEXT j
910 READ l$
920 POKE 23296,0
930 FOR j=1 TO LEN l$
940 POKE 23297,sy+100-8*j: POKE
23298,CODE l$(j): RANDOMIZE USR
start
950 NEXT j
2000 REM CHARACTER PLOTTER
2010 DATA 58,2,91,254,32,56,76,2
54,128,56
2020 DATA 10,254,144,56,68,254,1
65,56,9,24
2030 DATA 62,33,7,61,214,32,24,6
,42,123
2040 DATA 92,43,214,143,17,8,0,2
54,0,40
2050 DATA 4,25,61,24,248,6,0,126
,43,229
2060 DATA 14,0,24,1,241,7,245,19
7,245,33
2070 DATA 0,91,126,254,249,48,18
,129,79,35
2080 DATA 126,254,169,48,10,128,
71,24,8,24
2090 DATA 229,24,220,24,117,24,1
11,62,16,128
2100 DATA 71,33,0,64,254,128,48,
9,17,0
2110 DATA 8,25,254,64,48,1,25,20
3,184,203
2120 DATA 176,62,63,144,17,32,0,
254,8,56
2130 DATA 5,214,8,25,24,247,254,
0,40,4
2140 DATA 61,36,24,248,121,254,8
,56,5,214
2150 DATA 8,35,24,247,79,62,7,14
```

```
5,55,63
2160 DATA 87,71,94,254,0,40,4,20.
3,11,16
2170 DATA 252,241,56,4,203,131,2
4,2,203,195
2180 DATA 122,66,254,0,40,4,203,
3,16,252
2190 DATA 115,193,12,121,254,8,3
2,147,4,241
2200 DATA 225,120,254,8,32,141,4
0,4,241,193
2210 DATA 241,225,201
3000 REM HORIZONTAL LABELS
3010 DATA "JAN","FEB","MAR","APR
","MAY","JUN","JUL","AUG","SEP",
"OCT","NOV","DEC"
4000 REM VERTICAL LABELS
4010 DATA " 0"," 5","10","15","2
0","25","30","35","40","45","50"
4020 DATA "£1000s","BANK BALANCE
"
```

Program 10.2

Lines 500–590 plot the labels next to the graduations on the horizontal axis. These labels each contain three letters, and run vertically down the screen. This effect is achieved using two nested loops, *j* controlling the horizontal positioning, and *i* controlling the vertical movement at each horizontal position.

The vertical labels are plotted using a similar technique. Note that the vertical labels all contain two characters, allowing the use of a single positioning technique.

Once all the hard work has been done, and the axes produced, the graph may be drawn.

Use PLOT commands to produce a display showing individual values.

Use PLOT and DRAW to produce a continuous line graph.

If necessary, use a scale factor to place points correctly, according to the positions of the graduations and scales used on the axes. A scale factor can be used to expand or compress values in the x or y directions. Remember that a point which has co-ordinates x and y in relation to the origin of the graph must be placed correctly on the screen by taking account of the position of the origin of the graph in relation to the origin of the true screen co-ordinates. If the axes start at sx,sy then the point x,y must be plotted using screen co-ordinates $x+sx,y+sy$.

If a scale factor is used, the co-ordinates become

$x*xf+sx,y*yf+sy$

where xf and yf are the x and y scale factors.

Example 10.3

Add the following lines (Program 10.3) to the program given in Example 10.2.

```
1000 REM READ DATA INTO ARRAY
1010 DIM y(ng)
1020 FOR i=1 TO ng
1030 READ y(i)
1040 NEXT i
1100 REM PLOT POINTS ON GRAPH
1110 REM SCALE FACTORS xf,yf
1120 LET xf=1: LET yf=2
1130 FOR i=1 TO ng
1140 PLOT sx+xf*i*dg,sy+yf*y(i)
1150 NEXT i
5000 REM BANK BALANCE DATA
5010 DATA 15,18,12,14,20,35,44,5
0,30,26,32,45
```

Program 10.3

Line 5010 contains the bank balance, in £1000s. This data is read into an array, $y(1)$ to $y(12)$ in the section of the program from line 1000 to line 1040.

Once stored, the data is plotted using lines 1100 to 1150.

The x scale factor is 1 because each pixel in the x direction corresponds to a unit on the horizontal axis. However, in the y direction, the distance between the graduations is 10 pixels. This corresponds to only 5 units, so the scale is 2 pixels to 1 unit, i.e., a scale factor of 2 in the y direction.

Example 10.4

Instead of simply plotting the data as a series of points, replace lines 1130–1150, and add line 1160 as shown, in Program 10.4. This produces a jagged line graph, which is often a better way of illustrating a set of figures, as the direction of the graph is more easily seen than when points are used.

```
1130 PLOT sx+xf*dg,sy+yf*y(1)
1140 FOR i=2 TO ng
1150 DRAW dg*xf,yf*(y(i)-y(i-1))
1160 NEXT i
```
Program 10.4

Bar charts

Bar charts can be drawn on the screen using very similar techniques to those used for drawing graphs.

Axes are drawn in the same way as for line graphs, although the graduation marks do not normally project on either side of the horizontal axis, since this may interfere with the bottoms of the columns. This simply means that the graduations are produced by

PLOT *sx*+graduation number**dg,sy*
DRAW 0,*pp*

so that they project underneath the horizontal axis. The graduations may project on either side of the vertical axis, if there is a small gap before the first column begins to rise. Of course the graduations will have to be shortened if the first column rises close to the vertical axis.

The distance between horizontal graduations (*dg*) will depend on the width of each column, and the size of any gap between the columns, as shown in Fig. 10.5.

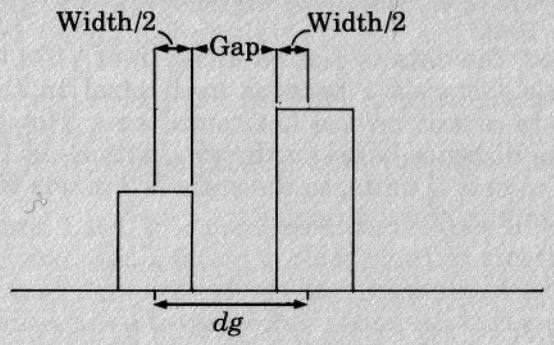


Figure 10.5 Important horizontal dimensions

The columns can be constructed in several ways. Unshaded columns can be produced by drawing their outlines using PLOT and DRAW commands.

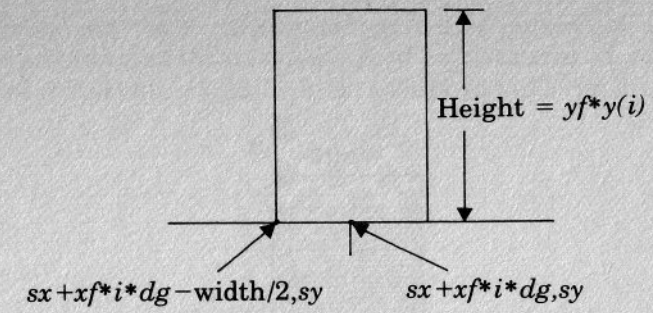


Figure 10.6 A column outline

Figure 10.6 shows the most important values when drawing a column outline. The central graduation will have co-ordinates

sx+*xf***i***dg,sy*

where *i* is the column number (or the graduation number). The start of the column will be half the column width nearer the origin, so this point will have co-ordinates

sx+*xf***i***dg*−width/2,*sy*

The height of column *i* will be *yf***y*(*i*).

Example 10.5

The data used in previous examples is used here to produce a bar chart using the same methods for constructing the axes. The main difference between the following program (10.5) and the program used in the previous example is to be found in lines 1100–1160. There are also minor differences in lines 330, 360, 380, 410 and 450. Make the following changes to the graph program developed so far:

```
  10 REM BANK BALANCE BAR CHART
 330 REM SPACED 16 APART
 360 LET sx=33: LET sy=33: LET w
idth=8
 380 LET dg=16
 410 PLOT sx,sy: DRAW ng*dg+widt
h/2,0
 450 DRAW 0,pp
1100 REM DRAW COLUMNS
1110 REM SCALE FACTORS xf,yf
1120 LET xf=1: LET yf=2
1130 FOR i=1 TO ng
1140 PLOT sx+xf*i*dg-width/2,sy
1150 DRAW 0,yf*y(i): DRAW width,
0: DRAW 0,-yf*y(i)
1160 NEXT i
```
Program 10.5

Instead of simply drawing the outlines of the columns, each column can be constructed by drawing small horizontal lines one on top of the other. This produces solid columns, as shown in Fig. 10.7.

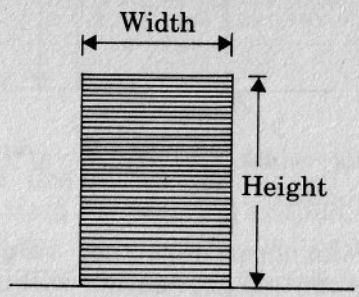


Figure 10.7 A solid column

Since the attributes are organized by print positions, columns of different colours can be constructed provided no two adjacent columns overlap the same print position. Normally, this can be arranged by ensuring that the columns are at least 1 print position, or 8 pixels, apart.

Each column can be drawn using a loop:

If the start of each column is denoted by *x*col,*y*col, as shown in Fig. 10.8.

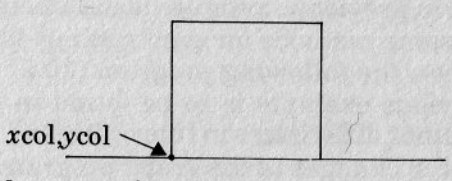


Figure 10.8 The column starting point

```
FOR i=0 TO height
PLOT xcol,ycol
DRAW width-1,0
NEXT i
```

Set the attributes for the print positions before drawing the columns, either by setting the attributes and clearing the screen, by using the ATTRIBUTE SET routine, by setting the appropriate attributes before drawing, or by employing temporary attributes inside PLOT or DRAW statements.

Shaded columns can be produced by drawing the outline of a column, then using either UDG characters or shape definitions to construct suitable shading.

A cross-hatched effect, as illustrated in Figure 10.9, is possible with a little care.

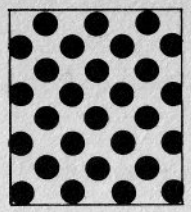


Figure 10.9 A cross-hatched column

The two characters shown in Fig. 10.10 are required to achieve this effect.

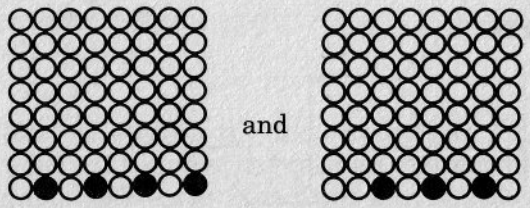


Figure 10.10 Characters used to produce cross-hatching

These characters can be overprinted alternately using the OVERPRINTER routine, moving the bottom of the characters up 1 pixel vertically each time.

An alternative technique is to use the CHARACTER PLOTTER routine to plot the shading characters, and draw the outline after the shading has been completed. If the outline is drawn first, the shading characters will obliterate part of the outline as they are drawn – if not the sides, then certainly the top of the column.

Cross-hatch shading can be constructed using only 1 UDG character if the bottom left corner of the character is moved in a zig-zag pattern.

Example 10.6

Make the following changes (Program 10.6) to the program developed in Example 10.5:

```
1050 REM READ COLOURS
1060 DIM c(ng)
1070 FOR i=1 TO ng
1080 READ c(i)
```

Program 10.6 *(continues)*

```
1090 NEXT i
1140 INK c(i)
1150 FOR j=1 TO yf*y(i)
1160 PLOT sx+xf*i*dg-width/2,sy+
j
1170 DRAW width-1,0
1180 NEXT j
1190 NEXT i
1200 INK 0
6000 REM COLUMN COLOURS
6010 DATA 1,2,3,4,5,6,0,1,2,3,4,
5
```

Program 10.6

Now, instead of drawing the outlines of the columns, the program draws a bar chart which has solid columns.

3D bar charts

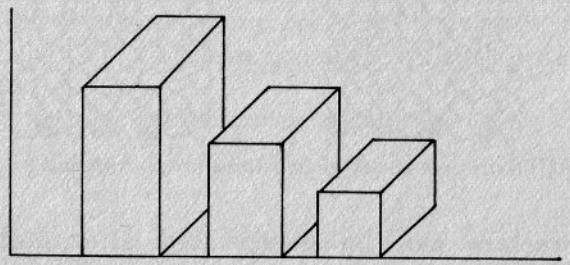


Figure 10.11 A 3D bar chart

A 3D effect can be added to a bar chart by drawing extra lines at the top and side of each column. Figure 10.11 shows what a 3D bar chart looks like, and Fig. 10.12 details the important values required for each column.

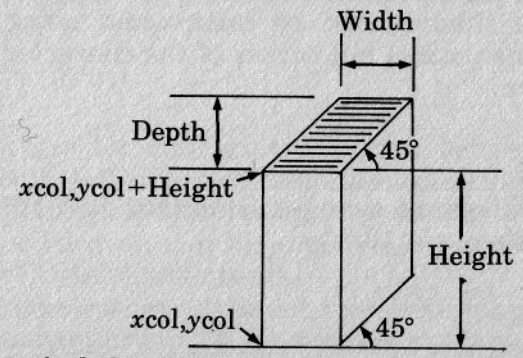


Figure 10.12 A typical 3D column

The top and side appear to lie at an angle to the front face of the column. Different angles produce a different illusion of depth, but 45° is convenient for most purposes. The top left corner will be

*x*col,*y*col+height

so use a loop to draw lines whose length = width − 1.

```
FOR i=2 TO depth
PLOT xcol+i,ycol+height+i
DRAW width-1,0
NEXT i
```

The effect of beginning the loop at 2 is to leave a blank line of pixels between the front and top faces of the column, simulating an edge.

The same process can be repeated for the side face.

```
FOR j=1 TO height - 1
PLOT xcol+width+2,ycol+j
DRAW depth-2,depth
NEXT j
```

Again, the effect of ending the loop at height − 1 produces the illusion of an edge at the top of the column, between the side and top faces. A vertical edge between the front and side faces is simulated by plotting the first point on the line at *x*col+width+2, i.e., 1 pixel farther to the right than expected.

Columns of different colours can be produced if care is taken to adjust the position of the origin of the chart (*sx,sy*) so that columns fall in print positions. Adjustment of the height of the horizontal axis is important if it is to be of a uniform colour and remain unaffected by the colours of the columns. The horizontal axis should be the top line of a print position so that its attributes are separate from those of the columns above.

Example 10.7

Delete lines 1100–1190 from the program used in the last example, and add the following (Program 10.7):

```
1100 REM DRAW COLUMNS
1110 REM SCALE FACTORS xf,yf
1120 LET xf=1: LET yf=2
1130 LET depth=6
1140 FOR i=1 TO ng
1150 INK c(i)
1160 LET xcol=sx+xf*i*dg-width/2
: LET ycol=sy: LET height=yf*y(i
)
```

Program 10.7 *(continues)*

```
1170 FOR j=1 TO yf*y(i)-1
1180 PLOT xcol,ycol+j
1190 DRAW width,0
1200 NEXT j
1210 REM 3D shading
1220 FOR j=1 TO depth
1230 PLOT xcol+j,ycol+height+j
1240 DRAW width-1,0
1250 NEXT j
1260 FOR j=1 TO height-1
1270 PLOT xcol+width+2,ycol+j
1280 DRAW depth-2,depth
1290 NEXT j
1300 NEXT i
1310 INK 0
```

Program 10.7

Pie charts

Pie charts can be constructed by placing radii around a circle. The angle between the radii should be in proportion to the fraction of the distribution which belongs to each sector.

For example, if there are 120 items in the data, and 20 of them are in one group, the angle between the radii forming that group is

$$\frac{20}{120} \times 2 \times \text{PI radians}$$

or, (number in group/total)*2*PI radians
Labels can be added to a pie chart if they are positioned carefully.

Example 10.8

A bird fancier who owns various colours of budgerigars decides to put a Spectrum to good use by drawing a pie chart showing the colours of the budgies in an aviary behind the house.

The following program (10.8) shows one way in which this task could be tackled.

```
 10 REM PIE CHART
 20 REM set colours
 30 PAPER 7: INK 0: BORDER 7
 40 CLS
100 REM set new ramtop
110 CLEAR 65164: REM 16K=32396
120 LET start=65165: REM 16K=32
397
```

Program 10.8 *(continues)*


```
200 REM place routine
210 LET length=203
220 LET s=start
230 FOR i=1 TO length
240 READ n
250 POKE s,n
260 LET s=s+1
270 NEXT i
300 REM initialise variables
310 READ n: REM no. of sectors
320 DIM t(n): REM no. in each s
ector
330 DIM c(n): REM cumulative to
tals
340 DIM a(n): REM cumulative an
gle
350 DIM b(n): REM extension ang
les
360 LET r=30: REM radius
370 REM define centre
380 LET sx=126: LET sy=88
390 LET e=10: REM extension
400 REM read no. in each sector
410 FOR i=1 TO n
420 READ t(i)
430 NEXT i
500 REM find totals and cumulat
ive totals
510 LET total=0
520 FOR i=1 TO n
530 LET c(i)=total+t(i)
540 LET total=total+t(i)
550 NEXT i
600 REM draw circle
610 CIRCLE sx,sy,r
700 REM cumulative angles
710 FOR i=1 TO n
720 LET a(i)=2*PI*c(i)/total
730 NEXT i
800 REM draw sectors
810 FOR i=1 TO n
820 GO SUB 4000
830 NEXT i
900 REM label extensions
910 FOR i=1 TO n
920 LET b(i)=2*PI*(c(i)-t(i)/2)
/total
930 PLOT sx+r*COS b(i),sy+r*SIN
b(i)
940 DRAW e*COS b(i),e*SIN b(i)
950 NEXT i
1000 REM plot labels
1010 FOR i=1 TO n
1020 READ l$
1030 IF b(i)>=0 AND b(i)<2*PI/4
```

```
THEN  GO SUB 5000
1040 IF b(i)>=2*PI/4 AND b(i)<PI·
 THEN  GO SUB 6000
1050 IF b(i)>=PI AND b(i)<2*PI*3
/4 THEN  GO SUB 7000
1060 IF b(i)>=2*PI*3/4 THEN  GO
SUB 8000
1070 POKE 23297,yl
1080 FOR j=1 TO LEN l$
1090 POKE 23296,xl+j*8-8
1100 POKE 23298,CODE l$(j)
1110 RANDOMIZE USR start
1120'NEXT j
1130 NEXT i
1200 REM print name
1210 READ n$
1220 PRINT AT 0,16-LEN n$/2;n$
1230 GO TO 9990: REM end
4000 REM blank sector
4010 PLOT sx,sy
4020 DRAW r*COS a(i),r*SIN a(i)
4030 RETURN
5000 REM angle in 1st quadrant
5010 LET xl=sx+(r+e)*COS b(i)
5020 LET yl=sy+(r+e)*SIN b(i)
5030 RETURN
6000 REM angle in 2nd quadrant
6010 LET xl=sx+(r+e)*COS b(i)-8*
LEN l$
6020 LET yl=sy+(r+e)*SIN b(i)
6030 RETURN
7000 REM angle in 3rd quadrant
7010 LET xl=sx+(r+e)*COS b(i)-8*
LEN l$
7020 LET yl=sy+(r+e)*SIN b(i)-8
7030 RETURN
8000 REM angle in 4th quadrant
8010 LET xl=sx+(r+e)*COS b(i)
8020 LET yl=sy+(r+e)*SIN b(i)-8
8030 RETURN
9000 REM CHARACTER PLOTTER
9010 DATA 58,2,91,254,32,56,76,2
54,128,56
9020 DATA 10,254,144,56,68,254,1
65,56,9,24
9030 DATA 62,33,7,61,214,32,24,6
,42,123
9040 DATA 92,43,214,143,17,8,0,2
54,0,40
9050 DATA 4,25,61,24,248,6,0,126
,43,229
9060 DATA 14,0,24,1,241,7,245,19
7,245,33
9070 DATA 0,91,126,254,249,48,18
,129,79,35
```



```
9080 DATA 126,254,169,48,10,128,
71,24,8,24
9090 DATA 229,24,220,24,117,24,1
11,62,16,128
9100 DATA 71,33,0,64,254,128,48,
9,17,0
9110 DATA 8,25,254,64,48,1,25,20
3,184,203
9120 DATA 176,62,63,144,17,32,0,
254,8,56
9130 DATA 5,214,8,25,24,247,254,
0,40,4
9140 DATA 61,36,24,248,121,254,8
,56,5,214
9150 DATA 8,35,24,247,79,62,7,14
5,55,63
9160 DATA 87,71,94,254,0,40,4,20
3,11,16
9170 DATA 252,241,56,4,203,131,2
4,2,203,195
9180 DATA 122,66,254,0,40,4,203,
3,16,252
9190 DATA 115,193,12,121,254,8,3
2,147,4,241
9200 DATA 225,120,254,8,32,141,4
0,4,241,193
9210 DATA 241,225,201
9500 REM NO. OF SECTORS
9510 DATA 5
9600 REM QUANTITIES IN SECTORS
9610 DATA 10,20,5,15,10
9700 REM SECTOR LABELS
9710 DATA "BLUE","GREEN","RED","
YELLOW","WHITE"
9800 REM PIE CHART LABEL
9810 DATA "BUDGIE COLOURS"
9990 REM finished
```

Program 10.8

The main stages in this program are:

Load the CHARACTER PLOTTER routine. This will be used to plot the labels showing what the various sectors represent.
Initialize a number of arrays and simple variables.
Read the number in each sector into an array, *t(i)*.
Find the total number of items to be included in the pie chart.
Calculate a set of cumulative totals which represent the number of items from the start of the distribution to the end of the current sector, e.g., if there are 10 items in sector 1 and 20 items in sector 2, the cumulative total to the end of sector 2 is 30 (i.e., 10 + 20).

A circle is drawn to produce the outline of the pie chart.
Using the cumulative totals held in $c(i)$, a set of angles is calculated, and placed in an array $a(i)$. The angle for each sector is the angle measured from the horizontal (the positive x direction), anticlockwise, to the end of the sector.
Radii are drawn at the angles held in the array $a(i)$, to form the sectors.
A short extension line is drawn from the circumference at the centre of each sector. This will connect each sector label to the appropriate sector. The length of this extension line is set in line 390.
The labels for the sectors are read and plotted using the CHARACTER PLOTTER routine. The bottom left co-ordinate of the first character in each label depends in which quarter (or Quadrant) of the circle the extension is drawn, as shown in Fig. 10.13.

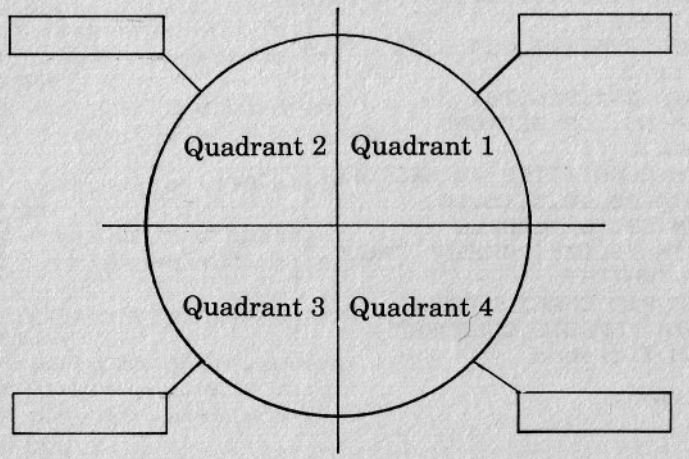


Figure 10.13 Relative positions of labels and quadrants

In quadrant 1:
The end of the extension represents the bottom left corner of the first character.
In quadrant 2:
The end of the extension represents the bottom right corner of the last character.
In quadrant 3:
The end of the extension represents the top right corner of the last character.

In quadrant 4:
The end of the extension represents the top left corner of the first character.
The subroutines beginning at lines 5000, 6000, 7000, and 8000 calculate the co-ordinates of the bottom left corner of the first character of label l$ assigning the result to xl and yl.
The labels are plotted.
A caption for the whole pie chart is read and printed, centred, at the top of the screen.

Solid shaded sectors can be added to the previous program using the simple technique involving the use of a large number of radii drawn one after the other. If this is to be done, the initial circle should not be drawn using the CIRCLE command, because this will result in a mis-match between the points generated by this command and the endpoints of the radii calculated using SIN and COS functions. While this is not noticeable when an individual radius is drawn, it tends to catch the eye when a whole shaded sector is drawn. Instead of the CIRCLE command, use the following routine.

When *sx,sy* denotes the centre of the circle,
and *r* denotes the radius:

```
LET x1=5: LET y1=0: REM initial point
PLOT sx+x1,sy+y1
FOR t=0 TO 2*PI+.1 STEP .1
LET x2=INT (r*COS t+.5)
LET y2=INT (r*SIN t+.5)
DRAW x2-x1,y2-y1
LET x1=x2
LET y1=y2
NEXT t
```

Shaded sectors should not be placed next to each other, so some means of shading (say) every second sector is required. This is easily arranged using the following method of identifying the even numbered sectors.

When i is the sector number:

```
IF i/2=INT (i/2) THEN use a solid sector
IF i/2<>INT (i/2) THEN use a blank sector
```

The corresponding subroutines which will produce shaded and unshaded sectors are:

```
REM shaded sector
PLOT sx,sy
DRAW r*COS a(i-1),r*SIN a(i-1)
FOR j=a(i-1) TO a(i) STEP 0.01
PLOT sx,sy
DRAW INT (r*COS j+.5),INT (r*SIN j+.5)
NEXT j
RETURN
REM unshaded sector
PLOT sx,sy
DRAW r*COS a(i),r*SIN a(i)
RETURN
```

Unfortunately, although the shaded sectors are normally quite solid looking, depending on the exact position and size of the sectors, and the radius of the circle, there may be occasional flaws. This is caused by the inbuilt SIN function and the algorithm used to draw lines in the Spectrum ROM. However, for most purposes the results are acceptable.

A different approach to the problem involves plotting individual points on a series of radii. This produces a solid shaded sector with no blemishes, provided the step size in the routine is suitable. However, because there are a large number of points to be plotted, all using BASIC, the shading does take some time to complete.

Example 10.9

Make the following substitutions and additions (Program 10.9) to the program given in Example 10.8.

```
 600 REM draw circle
 610 LET x1=r: LET y1=0: REM ini
tial point
 620 PLOT sx+x1,sy+y1
 630 FOR t=0 TO 2*PI+.1 STEP .1
 640 LET x2=INT (r*COS t+.5)
 650 LET y2=INT (r*SIN t+.5)
 660 DRAW x2-x1,y2-y1
 670 LET x1=x2
 680 LET y1=y2
 690 NEXT t
 800 REM draw sectors
 810 FOR i=1 TO n
 820 IF i/2=INT (i/2) THEN  GO S
UB 3000: REM solid sector
```



```
 830 IF i/2<>INT (i/2) THEN  GO
SUB 4000: REM blank sector
 840 NEXT i
3000 REM solid sector
3010 PLOT sx,sy
3020 DRAW r*COS a(i-1),r*SIN a(i
-1)
3030 FOR j=a(i-1) TO a(i) STEP 0
.01
3040 PLOT sx,sy
3050 DRAW INT (r*COS j+.5),INT (
r*SIN j+.5)
3060 NEXT j
3070 RETURN
```

Program 10.9

Try altering the step size in line 3030, and watch the effect on the appearance of the solid sectors.

Try changing the data too, in lines 9510 and 9610.

11 DATASHOW

DATASHOW is a short program which displays information entered at the keyboard, on a graph.
The program (11.1) allows the following items to be input:

(a) The number of labels on the horizontal axis, i.e., the number of horizontal graduations.
(b) The horizontal labels, i.e., the name attached to each horizontal graduation.
(c) The name attached to the vertical axis.
(d) The name attached to the horizontal axis.
(e) The vertical values, i.e., the points to be plotted.

A simple form of automatic scaling is used on the vertical axis, to allow several ranges of values to be displayed.

```
  10 REM DATASHOW
  20 REM set colours
  30 INK 0: PAPER 7: BORDER 7
  40 CLS
 100 DIM l$(12,3): DIM v(12)
 200 REM INPUT NO. OF LABELS
 210 CLS : PRINT "How many horiz
ontal labels?"
 220 INPUT l
 300 REM INPUT HORIZONTAL LABELS
 310 FOR i=1 TO l
 320 CLS : PRINT "Input label ";
i
 330 INPUT l$(i)
 340 NEXT i
 400 REM NAME THE VERTICAL AXIS
 410 CLS : PRINT "Name the verti
cal axis"
 420 INPUT v$
 450 REM NAME THE HORIZONTAL AXI
S
 460 CLS : PRINT "Name the horiz
ontal axis"
 470 INPUT h$
 500 REM INPUT VERTICAL VALUES
 510 FOR i=1 TO l
 520 CLS : PRINT "Input the valu
e for ";l$(i)
 530 INPUT v(i)
 540 NEXT i
 600 REM FIND MAX & MIN VERTICAL
```

Program 11.1 *(continues)*


```
 VALUES
 610 LET max=v(1): LET min=v(1)
 620 FOR i=1 TO l
 630 IF v(i)>max THEN  LET max=v
(i)
 640 IF v(i)<min THEN  LET min=v
(i)
 650 NEXT i
 700 REM FIX VERTICAL SCALE
 710 IF max>3000 THEN  PRINT "Va
lues too large": STOP
 720 LET m=INT ((LN max)/(LN 10)
)-1
 730 IF max<=60*10^m THEN  LET d
v=10*10^m
 740 IF max<=30*10^m THEN  LET d
v=5*10^m
 750 IF max<=12*10^m THEN  LET d
v=2*10^m
 760 CLS
 800 REM DRAW HORIZONTAL AXIS
 810 LET sx=44: LET sy=44
 820 LET ng=l: LET dg=16: LET pp
=2
 830 PLOT sx,sy: DRAW ng*dg,0
 840 FOR n=0 TO ng
 850 PLOT sx+n*dg,sy-pp
 860 DRAW 0,pp+pp
 870 NEXT n
 880 FOR i=1 TO l
 890 FOR j=1 TO LEN (l$(i))
 900 PRINT AT 16+j,5+INT (dg/8)+
2*(i-i);l$(i,j)
 910 NEXT j
 920 NEXT i
1000 REM DRAW VERTICAL AXIS
1010 LET yng=6: LET ydg=16
1020 PLOT sx,sy: DRAW 0,yng*ydg
1030 FOR n=0 TO yng
1040 PLOT sx-pp,sy+n*ydg
1050 DRAW pp+pp,0
1060 NEXT n
1100 FOR i=0 TO yng
1110 PRINT AT 16-2*i,5-LEN (STR$
 (i*dv));i*dv
1120 NEXT i
1200 REM ADD VERTICAL NAME
1210 FOR i=1 TO LEN (v$)
1220 PRINT AT 10-INT (LEN (v$)/2
)+i-1,0;v$(i)
1230 NEXT i
1250 REM ADD HORIZONTAL NAME
1260 PRINT AT 21,5+INT ((ng*dg/8
)/2)-INT (LEN h$/2);h$
1300 REM PLOT POINTS
```

```
1310 PLOT sx+dg,sy+v(1)*ydg/dv
1320 FOR i=2 TO l
1330 DRAW dg,(v(i)-v(i-1))*ydg/d
v
1340 NEXT i
```

Program 11.1

Program description

10–40		Program title
100		Dimension arrays l$(12,3) represents the horizontal labels v(12) represents the vertical values
200–210		Input l, the number of horizontal graduations.
300–340		Input the label for each horizontal graduation.
400–410		Input the name of the vertical axis.
450–470		Input the name of the horizontal axis.
500–540		Input the vertical values, i.e., the height of each point on the graph.
600–650		Find the maximum and minimum vertical values. The maximum value will determine the scale used on the vertical axis. The minimum value could be used to determine the smallest value which appears on the vertical axis, although it is not used for this purpose in this program.
700–790		Determine which scale is to be used on the vertical axis.
	710	Stop, after printing an error message, if max>6000.
	720	If max is expressed in the form number × 10 ↑ m *m*, the power of ten, can be found from m = INT(LOG m) − 1 i.e., *m* is the characteristic of the logarithm of *m*. The Spectrum does not have a $\log_{10}$ function, so this must be synthesized using natural logarithms, where

$$\text{LOG}_{10}\ m = \frac{\text{LN } m}{\text{LN } 10}$$

	730–750	The basic scales available use intervals of 2, 5, or 10 on the vertical axis. Six intervals are used, so that the maximum values on these scales are 12, 30, and 60. The basic scales are modified by multiplying by 10↑*m*, to produce other, larger scales. The interval which is to be used is selected so that the maximum value will fit on the graph.
	760	Clear the screen before drawing the graph.
800–900		Draw the horizontal axis.
	810	The origin of the axes is set at 44,44
	820	The number of graduations (*ng*) is the same as the number of horizontal labels (*l*). The distance between graduations (*dg*) is 16 pixels. The pixel projection (*pp*) represents the distance by which the graduations project on either side of the axis.
	830	Plot the origin, to establish the left end of the axis, then draw a horizontal line of length No. of graduations × distance between graduations
	840–870	Draw each graduation.
	880–920	Print each label vertically, below the appropriate graduation on the axis.
1000–1120		Draw the vertical axis.
	1010	The number of graduations (*yng*) is six. The distance between graduations (*ydg*) is 16 pixels.
	1020	Plot the origin, to establish the bottom of the axis, then draw a vertical line of length No. of graduations × distance between graduations
	1030–1060	Draw each graduation.
	1100–1120	Print the vertical labels. The value of each label is graduation number × scaled distance between graduations

1200–1230 Print the name of the vertical axis, vertically. Printing begins at line
10 − ½(length of label)
and progresses downwards.

1250–1260 Print the name of the horizontal axis, horizontally. Printing begins at column
5 + ½(length of axis) − ½(length of label)
and takes place from left to right.

1300–1340 Plot and join the points on the graph.

1310 Plot the first point above the first graduation on the horizontal axis. The height above the axis is

$$\frac{\text{vertical value}}{\text{value of distance between graduations}} * \text{No. of pixels between each graduation}$$

The value of the expression

$$\frac{\text{vertical value}}{\text{value of distance between graduations}}$$

represents the height above the axis, expressed as a number of graduations.
Multiply the number of graduations by the number of pixels between each graduation, to find the height above the axis as a number of pixels.

1320–1340 Draw from the previous point to the current point. The horizontal distance is the distance between graduations.
The vertical distance is found by subtracting heights, and converting to a number of pixels.

APPENDIX A

Page boundaries

This appendix contains the address of the first byte on each of a number of pages of memory. In general, when creating a second screen, choose the highest available address which allows room for the screen and any other routines which are required.
On a 16K machine, select an address which is less than

(32599 − file length − routines).

65280	65024	64768	64512	64256	64000	63744
63488	63232	62976	62720	62464	62208	61952
61696	61440	61184	60928	60672	60416	60160
59904	59648	59392	59136	58880	58624	58368
58112	57856	57600	57344	57088	56832	56576
56320	56064	55808	55552	55296	55040	54784
54528	54272	54016	53760	53504	53248	52992
52736	52480	52224	51968	51712	51456	51200
50944	50688	50432	50176	49920	49664	49408
49152	48896	48640	48384	48128	47872	47616
47360	47104	46848	46592	46336	46080	45824
45568	45312	45056	44800	44544	44288	44032
43776	43520	43264	43008	42752	42496	42240
41984	41728	41472	41216	40960	40704	40448
40192	39936	39680	39424	39168	38912	38656

38400	38144	37888	37632	37376	37120	36864
36608	36352	36096	35840	35584	35328	35072
34816	34560	34304	34048	33792	33536	33280
33024	32768	32512	32256	32000	31744	31488
31232	30976	30720	30464	30208	29952	29696
29440	29184	28928	28672	28416	28160	27904
27648	27392	27136	26880	26624	26368	26112

APPENDIX B

Multiple screens on the 16K Spectrum

Every extra display file requires 6144 bytes of storage space, so if an extra display file is stored in memory there is very little space left for a BASIC program, workspace, or machine code routines. There is room to store one extra display file and one extra attribute file in the 16K Spectrum, but BASIC programs must be written in as compact a form as possible.

Many of the example programs listed in Chapter 6 will not run on the 16K machine unless they are modified.

Several steps may be taken, to reduce the length of a BASIC program

1. Use a separate program to place the machine code routines in memory. These routines can then be saved as CODE and loaded directly into memory from the main program. There are examples of this technique in Chapters 5 and 9.
2. Complex screen displays may be constructed using separate programs, and saved using SCREEN$ or CODE.
   These screens may then be loaded from the main program as necessary. (See the note at the end of Chapter 7.)
3. Remaining commands should be written using multiple statements on each line, and unnecessary REM statements should be omitted.

A suitable address for the start of a second screen in the 16K model is 25600, and this will place the start of the second attribute file at 31744.

Despite the fact that many of the examples in Chapter 7 require modification if they are to run on 16K Spectrums, the techniques remain valid, and are capable of producing extremely high quality graphics, particularly where shapes are made to move in front of a background.

INDEX

A guide, to help you find the routines in this book.

Advanced graphics

As a Spectrum user, you will have seen the kind of high quality graphics that can be produced on this machine. But do you know how to do it yourself?

Two complete systems

Two complete graphics systems are described and listed in this book: *character graphics*, and a powerful system based on *shape tables.* Much of the writing is in machine code, but you are given complete routines and do not need to understand machine code to use the systems. They enable you to create fast, high-quality graphics quickly and easily.

And there is much, much more: three complete games (Dragon's Mouth, Dominoes, and Saturn's Surface), techniques for multiple screens, arcade graphics, and business graphics (graphs, bar charts and pie charts).

A step forward

This book does *not* deal with the elementary graphics described in the Spectrum Manual. It does *not* teach you BASIC. But it *does* teach you, in a simple and enjoyable way, how to produce professional graphics on the Spectrum, and it *does* provide the sophisticated software that is an essential tool for the programmer.

McGRAW-HILL Book Company (UK) Limited
MAIDENHEAD · BERKSHIRE · ENGLAND 07 084747 9